Optimal Control of Nonlinear Plants using Artificial Neural Networks

by

Mansour Abdulaziz Al-Dajani

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

SYSTEMS ENGINEERING

June, 1997

Optimal Control of Nonlinear Plants Using Artificial Neural Networks

BY

MANSOUR ABDULAZIZ AL DAJANI

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

SYSTEMS ENGINEERING

June, 97

KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS
DHAHRAN 31261, SAUDI ARABIA

COLLEGE OF GRADUATE STUDIES

This thesis, written by Mansour Abdulaziz Al Dajani under the direction of his Thesis Advisor and approved by his thesis Committee, has been presented to and accepted by the dean of the College of Graduate Studies, in partial fulfillment of the requirements for the degree of MASTER OF SCIENCE IN SYSTEMS ENGINEERING.

Thesis Committee

Md. Shafqir Ahmed
Thesis Advisor (Dr. M. S. Ahmed)

Member (Dr. Davut Kavranoglu)

Member (Dr. Fouad M. AL Sunni)

H. Andijani
Department Chairman

M. Elshafei
Member (Dr. M. El Shafei Ahmed)

Dean, Collage of Graduate Studies

Date: 15 June, 1997

كلية الدراسات العليا
COLLEGE OF GRADUATE STUDIES

Thesis Abstract

Name of Student:	Mansour Abdulaziz Al Dajani
Title of Study:	Optimal Control of Nonlinear Plants Using Artificial Neural Networks.
Major Field:	Systems Engineering
Date of Degree:	June, 1997

In this thesis, algorithms for application of Artificial Neural Networks in solving nonlinear optimal control problems are developed. The conventional Multi-layers Feed-forward Neural Network is employed as the state feedback optimal controller. The newly developed Block Partial Derivatives concept is used to compute the gradient needed for neural network training. State tracking, regulation, terminal control, minimum control effort, minimum time, and output tracking problems are attempted. The performance of the proposed algorithms is investigated through application on simulated plants. Results obtained agree with the ones found through other standard techniques.

MASTER OF SCIENCE DEGREE

KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS

Dhahran, Saudi Arabia

June, 1997

خلاصة الرسالة

اسم الـــطالب : منصور عبدالعزيز عبدالرحيم الدعجاني

عنوان الدراسـة : التحكم الأمثل في الأنظمة اللاخطية باستخدام الشبكات العصبية الاصطناعية.

التخـــصـص : هندسة النظم

تاريخ الشـــهادة : صفر ١٤١٨ هـ

في هذه الرسالة، تم تطوير خوارزميات جديدة لتطبيقات الشبكات العصبية الاصطناعية لحل مسائل التحكم الأمثل اللاخطية. وقد استخدِمت شبكات عصبية من النوع المتعدد الطبقات بتغذية أمامية على شكل متحكِّم أمثل في نظام تغذية خلفية مرتدّة لمتغيرات الحالة. وقد استخدِمت الطريقة المطورة حديثاً والمسماة التفاضل الجزئي للمجموعات وذلك لحساب درجة التحدّر اللازمة لتدريب الشبكة العصبية. وقد شملت الدراسة عمليات تعقب متغيرات الحالة، تنظيم قيم المتغيرات، التحكم الطرفي للمتغيرات، التحكم بأقل جهد، التحكم بأقل وقت، وأخيراً تعقب خرج المحكومة. وقد قُيم أداء الخوارزميات المطورة بتطبيقها على بعض من مسائل التحكم الأمثل. وقد وجد أن النتائج المستخلصة تتوافق مع حلول تلك المسائل بالطرق الرقمية الأخرى.

درجة الماجستير في العلوم

جامعة الملك فهد للبترول والمعادن

الظهران، المملكة العربية السعودية

صفر ١٤١٨ هـ

dedicated to my parents.

ACKNOWLEDGMENT

Acknowledgment is due to King Fahd University of Petroleum and Minerals for the support of this research.

I wish to express my deep appreciation to my major thesis advisor Professor Mohammed Shahgir Ahmed for his guidance, support, and continuous assistance. I also wish to thank my thesis committee members Dr. Davut Kavranoglu, Dr. Fouad AL-Sunni, and Dr. Moustafa El Shafei Ahmed for their helpful comments.

Table of Contents

List of Tables

List of Figures

CHAPTER I

INTRODUCTION

1.1 General Overview

Apart from the classical techniques for design of control systems, more advanced techniques are being developed to deal with increasingly complex systems to meet stringent design requirements and to accommodate lack of knowledge about the plants to be controlled. Neural networks are found to be successful in controlling dynamic systems that were otherwise difficult or impossible to control using the tools provided by the classical control. During the last few decades, many algorithms for adapting the neural networks to control dynamic systems have been developed. Dynamic Backpropagation (DB) [27, 28] and Backpropagation Through Time (BPTT) [45] are examples of such algorithm. With the aid of these algorithms, neural networks have been found to be promising in implementing adaptive, optimal, and nonlinear control in different types of plants [28, 5, 16, 29, 21].

Most of the available techniques for solving the nonlinear optimal control problem are based on numerical techniques. Further, they provide the solution in

an open-loop fashion. The optimal control signal is obtained off-line and then applied to the system during implementation. From practical point of view, open loop control is highly undesirable. This stems from the fact that any change in the initial states of the system or perturbation in the system parameters may cause the actual trajectory of the system to diverge from the optimal one and may even make the system unstable. Artificial Neural Networks (ANN's) have emerged to provide closed loop solution to a wide class of optimal control problems as will be elaborated in chapter II.

1.2 Thesis Objective

The training of an ANN to implement optimal control strategies usually involves the computation of a gradient which reflects the change of a performance index with respect to the change in the parameters of the network. The existing algorithms for gradient computation like DB and BPTT can be only used for very simple recurrent networks. The derivation becomes inconceivable even in the presence of a modest number of feedback loops in the network. In a control configuration, there may exist many feedback loops and complicated interconnections. However, the recently developed Block Partial Derivatives (BPD) concept is applicable for gradient computations in networks with arbitrary interconnections. In this scheme, there exists virtually no restrictions on the interconnection. Thus, it is believed that the BPD concept can be successfully exploited in solving most optimal control problems.

The objective of this thesis is to utilize the BPD concept in developing simple and direct algorithms for neural net based state feedback optimal control of nonlinear plants.

1.3 Scope of the Thesis

In this thesis, design of neural net based state feedback optimal controllers are considered. The conventional Multi-layers Feed-forward Neural Network is employed as the controller. State tracking, regulation, terminal control, minimum control effort, minimum time, and output tracking problems are considered.

The thesis is organized as follows. The following sections of chapter I provide an overview of the ANN's, their definitions, structures, types, and learning algorithms. Moreover, the optimal control problems are defined and the solution of the general discrete optimal control problem is presented. Chapter II is a brief literature survey on previous research work in the area. The development of the neural net state feedback optimal control is presented in chapter III. These control schemes are validated through simulation in chapter IV. Chapter V presents conclusions and recommendations for further work in this field.

1.4 Artificial Neural Networks

1.4.1 Definition of the Neural Network

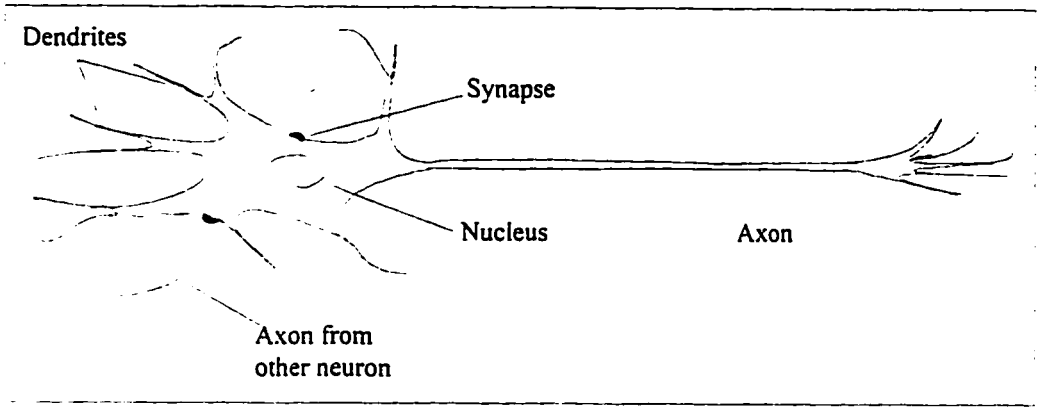


Figure 1.1 Elementary Nervous Cell.

An artificial neural network is a massively parallel distributed processor that has a natural propensity of storing experiential knowledge and making it available for use. They work in an analogs way to the Neurobiological system in the human brain. The human brain has an estimated number of 10 billion neurons constituting a highly efficient structure [13]. The neurons in the brain are the central data processing units. They are connected to each other by synaptic connections called synapses. Figure 1.1 shows an elementary nervous cell known as the pyramidal cell. The axons and the dendrites are, respectively, the transmission and reception media for the cell.

Artificial Neural Networks (ANN) mimic the nervous cells in their basic structure and functionality. ANN consist of artificial neurons connected to each

other using artificial synoptic connectors. ANN are built using electronic hardware components or simulated using digital computers. ANN have been used widely in solving engineering problems in various topics like control, signal processing, and pattern recognition. Like the brain, ANN's have the ability to learn by changing their structure in order to achieve certain objective. The networks can then be used to make decisions for situations laying in the training domain.

1.4.2 Structure of the Artificial Neural network

The neuron is the main component of the ANN. Figure 1.2 shows the basic components of the neuron. The inputs to the neuron are premultiplied by the synaptic weights. The resulting signals are summed up using a summing junction. $\Gamma(s)$ is the activation function which is also called the squashing function. Its purpose is to limit the output of the neuron in a desired range.

There are many types of activation functions used in the literature such as the threshold, piecewise-linear and sigmoid functions. The later is the most commonly used for its smoothness and asymptotic properties [13]. The logistic function is one type of the sigmoid functions having the form,

$$y(s) = \frac{1}{1+e^{-a\,s}} \tag{1.1}$$

where a is a slope parameter. Another common type of sigmoid functions is the hyperbolic tangent function defined as,

$$y(s) = \tanh(s/2) = \frac{1-e^{-s}}{1+e^{-s}} \tag{1.2}$$

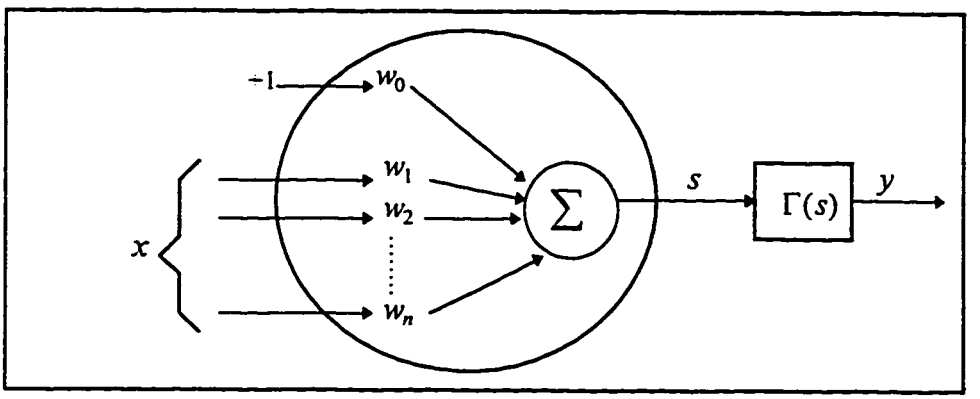


Figure 1.2 Artificial Neuron.

The hyperbolic tangent function can generate positive as well as negative output making it suitable for certain applications like control.

1.4.3 Types of Neural Networks

The structure of the interconnection between the neurons in the ANN characterizes its type. There are mainly three different classes of ANN, namely feed-forward, recurrent, and lattice structure networks [13].

Feed-forward Neural Networks: The neurons are arranged in layers as shown in Figure 1.3. Each layer consists of number of neurons and the output of each neuron is fed to the input of all or some of neurons in the following layer. The first and last layers are denoted as the input and output layers respectively. The intermediate layers are known as the hidden layers.

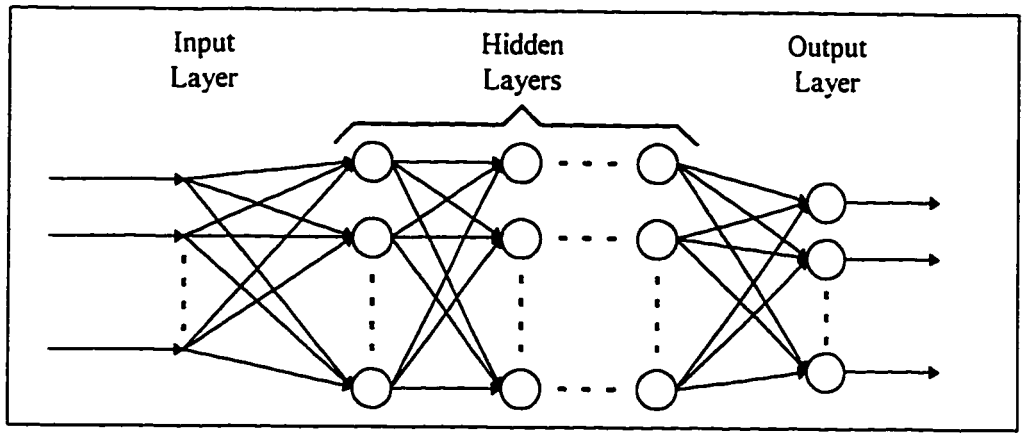


Figure 1.3 Multi-layers Feed-forward Neural Network

Recurrent Neural Networks: Recurrent networks are characterized by feedback loops going from one or more neurons to other in the same or previous layers.

Lattice Structure Neural Networks: In this type of networks, the neurons are arranged in a two (or more) dimensional array. The input source nodes are connected to each neuron in the array. Figure 1.4 shows a two dimensional lattice with two-by-two neurons collecting data from a common source node.

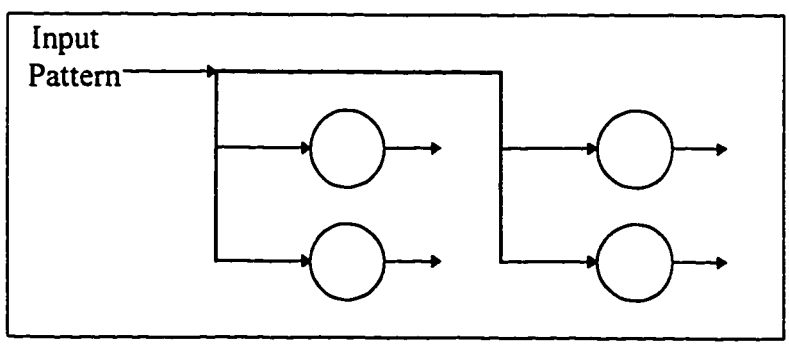


Figure 1.4 Two-by-two Lattice Network.

1.4.3 Learning Algorithms for Neural Networks

Training the neural network is the process of changing the network's free parameters (usually the synaptic weights) to minimize certain criterion. Many learning rules have been developed to train the ANN. These rules may be classified in three basic classes:[17], supervised, unsupervised, and hybrid learning.

As the name indicates, supervised learning is performed with the supervision of a "teacher". The basic structure of this training class is shown in Figure 1.5.

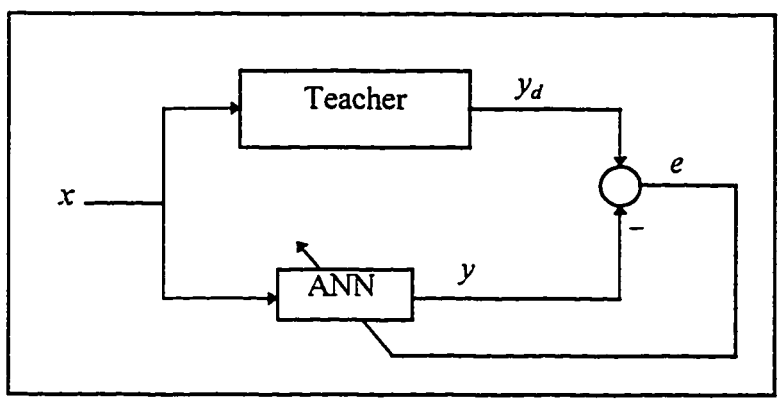


Figure 1.5 Supervised Training.

The network is trained in order to mimic the response of the "teacher" towards the received input data. One important training algorithm belonging to this class is known as the Backpropagation (BP) that is used for training feed-forward networks. BP will be discussed in more details in a separate subsection in this thesis.

The unsupervised learning is done without a teacher. It does not require the desired output y_d to be known during the training. It exploits the underlying

structure in the data, or correlation between patterns in the data, and organizes patterns into different categories from this correlation. Hybrid learning is the combination between the supervised and the unsupervised training.

1.4.4 Backpropagation Algorithm

Backpropagation (BP) is a simple minded gradient algorithm. BP is normally referred to the systematic gradient computation in the ANN that is derived using the chain rule of derivative computation. The algorithm is described here using an example of two-layer feed forward network as shown in Figure 1.6. The solid lines indicate the forward path and the dashed lines indicate the backward bath.

The weights are usually set initially to arbitrary small values. The algorithm starts by sweeping the input vector **X** to the network to generate the output vector **Y** thorough a normal feed forward process. The blocks *sgm*(.) and *sgm*'(.) indicate respectively the sigmoid function and its derivative. The outputs response vector Y is compared with the desired output vector **D**. The error vector in the output layer is given by,

$$\in = D - Y \tag{1.3}$$

The *sum square error* is then defined as,

$$\varepsilon^2 = \sum_{i=1}^{N_y} \in_{ik}^2 \tag{1.4}$$

where N_y is the number of neurons in the output layer and k is the time index. For the example network in Figure 1.6, the sum square error will be,

$$\varepsilon^2 = (d_1 - y_1)^2 + (d_2 - y_2)^2 \tag{1.5}$$

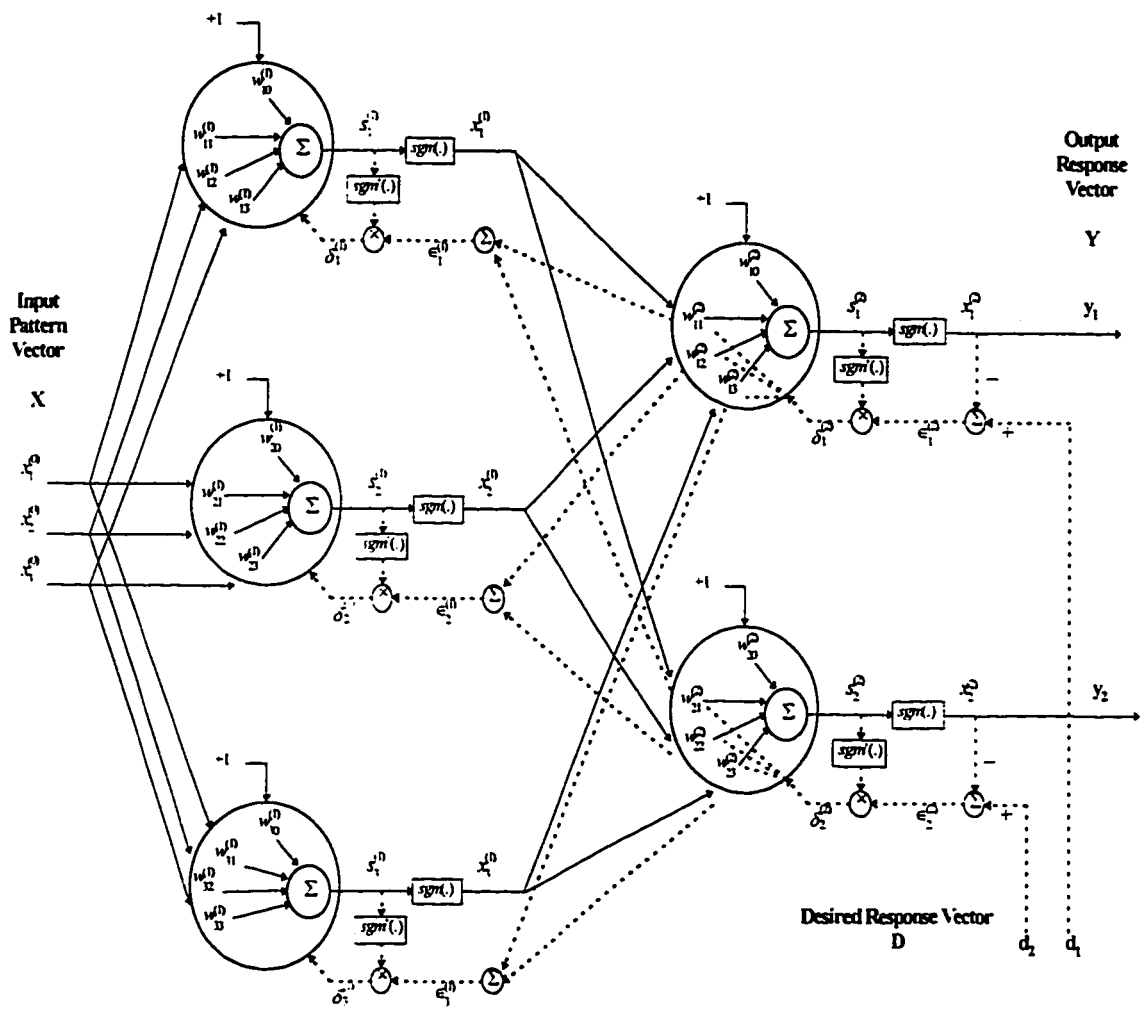


Figure 1.6 Example of two-layer back-propagation network architecture

The squared error is swept backward through the network. The *squared error derivative* δ for layer l and neuron j is defined as,

$$\delta_j^{(l)} = -\frac{1}{2}\frac{\partial \varepsilon^2}{\partial s_j^{(l)}} \tag{1.6}$$

where s is the output from the summing junction of the neuron. It is shown [47] that the error derivative is equal to the back-propagated error multiplied by the time-derivative of the sigmoid function evaluated at $s_j^{(l)}$, i.e.,

$$\delta_j^{(l)} = \in_j^{(l)} sgm'(s_j^{(l)}) \tag{1.7}$$

The errors in the first and hidden layers of the network are obtained in different way than the output layer. For the example shown in the figure, $\in_1^{(1)}$ is obtained from,

$$\in_1^{(1)} = \delta_1^{(2)} w_{11}^{(2)} + \delta_2^{(2)} w_{21}^{(2)} \tag{1.8}$$

The weights of the network are adjusted along the direction negative to the error gradient associated with their neurons. The gradient $\nabla_j^{(l)}$ corresponds to the jth neuron in layer l is defined as the partial derivative of the square error with respect to the change in weights of that neuron. In other words,

$$\nabla_j^{(l)} = \frac{1}{2}\frac{\partial \varepsilon^2}{\partial W_k} \tag{1.9}$$

Going through some analytical derivations [47], the gradient can be written as,

$$\nabla_k = -\delta_k X_k^{(l)} \tag{1.10}$$

where $X_k^{(l)}$ is the input vector to the neuron. Finally, the network weights are updated according to,

$$W_{k+1} = W_k + \eta\ (-\nabla_k) \tag{1.11}$$

where η is the adaptation rate of the weights. The steps above are repeated for all available data sets until the weights converge to their steady state values and the

sum square error attains its minimum value, where we claim that the network has been trained.

1.4.5 The Block Partial Derivatives (BPD) Concept

Block Partial Derivative (BPD) is a recently developed technique [3] for computation of the gradient vector in complex networks.

Definition: Consider a block A as shown in Figure 1.7a with the input $x \in R^{n_x}$ and the output $y \in R^{n_y}$. The block partial derivatives with respect to the block A are defined as,

$$\frac{\partial^A y_i}{\partial x_j} \equiv \lim_{\Delta x_j \to 0} \frac{\Delta y_i}{\Delta x_j}\Big|_{\Delta x_k = 0 \text{ for } k \neq j} \tag{1.12}$$

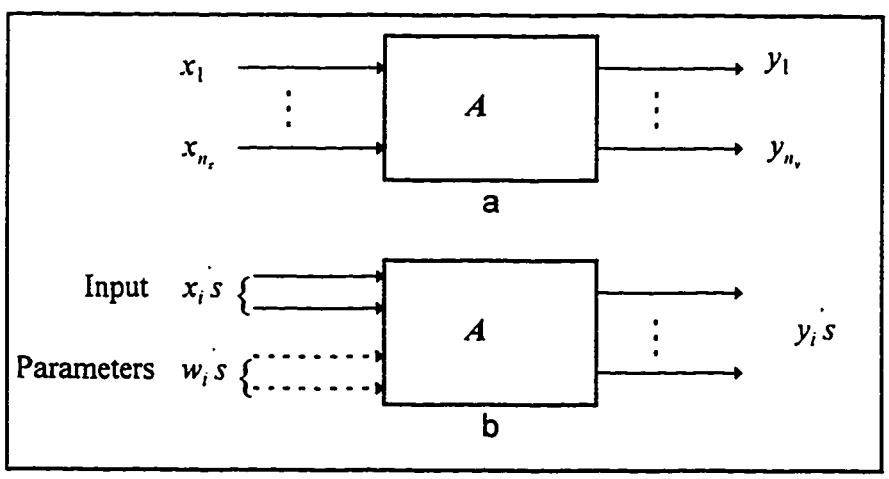


Figure 1.7 Blocks **a.** Block with input and output **b.** Parameters of the block shown as auxiliary input.

Remark 1: In the above definition all inputs other than x_j are held constant. However, the other signals within the block are allowed to vary owing to the change in x_j .

Remark 2: If a parameter set $w \in R^{n_w}$ in the block also is allowed to vary, its effect may be accommodated by representing the weights as auxiliary inputs as shown in Figure 1.7.b. This representation facilitates computation of BPD's with respect to the parameters.

Remark 3: When x and y are functions of time and the block is dynamic (i.e. has memory), the BPD's may assume rational dynamic operators [4].

Remark 4: The BPD's are similar to the ordinary partial derivatives except that a block (which may be imaginary) characterizes the explicit dependence of a variable on other variables. The 'other' variables are analogous to the independent variables in the ordinary partial derivatives. Thus, the algebra of ordinary partial derivatives can be easily modified for the BPD's.

BPD Computation: In [3], the algebra of block partial derivatives is presented that can be used to derive BPD's when various blocks are connected in different configurations. In the following, an alternate but simpler way of computing them is presented that is based on [4].

Consider the block labeled as A in Figure 1.8. The imaginary block A is made up of interconnections of various other blocks as shown in the figure. These blocks may be linear as well as nonlinear and static as well as dynamic. They may be simple arithmetic operators, real functions, or composite blocks having more blocks within. Further, they may have multiple input-output.

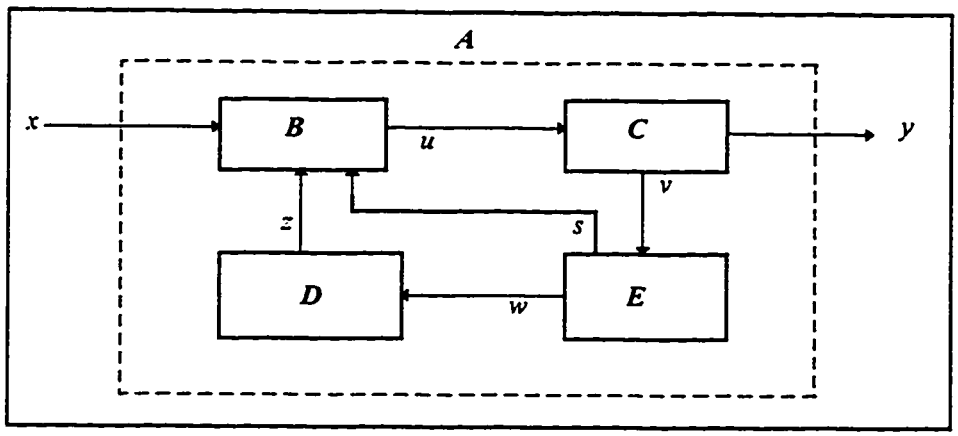


Figure 1.8 Computation of block partial derivatives (Block A made up of interconnections).

At this point, we are addressing the problem of finding the BPD's for the block A in terms of the BPD's of the other blocks.

Principle of the Argument: Since the definition of BPD involves infinitesimal change in one of the variables, it causes infinitesimal changes among all other variables inside the block. Thus every block inside A is also subjected to an infinitesimal change in its own input signals. Therefore, the change in their output due to the change in their own input can be represented as linear combinations of the input changes in terms of their own BPD's.

For example, an infinitesimal change in the output of block B can be written as,

$$\partial u = \frac{\partial^B u}{\partial x}\partial x + \frac{\partial^B u}{\partial s}\partial s + \frac{\partial^B u}{\partial z}\partial z \qquad (1.13)$$

The above suggests that in order to compute the BPD's for the block A, each block in Figure 1.8 can be replaced by a set of linear blocks each depicting the BPD corresponding to each of its own input-output pairs. The resulting 'linearized network' is shown in Figure 1.9. The BPD $\partial^A y/\partial x$ can be obtained by computing the graph transmittance from the node labeled ∂x to the node labeled ∂y. This is equivalent to feeding a unity signal at the node labeled ∂x of the 'linearized network' and measure the signal at the output node labeled ∂y. The linearized

network may be reduced using the classical Block diagram or graph reduction techniques [33]. For more details and examples the reader is referred to [3].

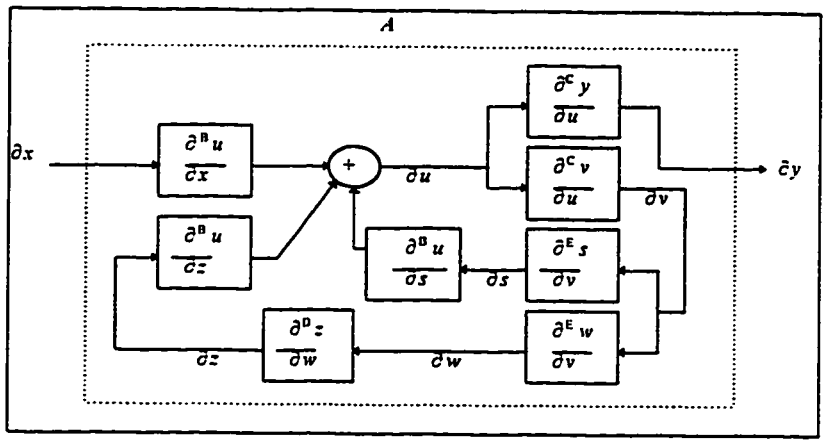


Figure 1.9 Linearized diagram for BPD computation.

Using the BPD concept to MFNN, the backpropagation algorithm can be rederived. The derivation does not require the chain rule of derivative computation. In fact, for an ordered (such as an MFNN) neural network, the BPD's are exactly equivalent to the ordered partial derivatives (OPD) [39].

1.5 Introduction to Optimal Control

1.5.1 Definition and Problem Formulation

Optimal control is concerned with the determination of the control signal that will cause the process to satisfy the physical constraints and at the same time minimize (or maximize) some performance criterion [19]. An optimal control

problem is formulated using the following two attributes:

- Dynamic equations that describe the behavior of the system states as a function of time for a given class of input.
- Performance index (PI) which is to be minimized (or maximized) by the optimal controller.

Thus one may define a discrete time optimal control problem in the following form [22],

Minimize the PI,

$$\Im = h(x_N, N) + \sum_{k=i}^{N-1} J^k(x_k, u_k) \tag{1.14}$$

subject to the discrete-time system dynamics,

$$x_{k+1} = f^k(x_k, u_k) \tag{1.15}$$

where x_k is an n-state vector and u_k is an m-control vector. The indices i and N are the initial and final time steps respectively.

The objective of an optimal control problem usually is to find the control signal u_k^* that will drive the system along a trajectory x_k^* such that the criterion function is minimized [22].

1.5.2 Solution to the General Discrete Optimal Control Problem

To solve the optimal control problem given by (1.14) and (1.15), the constraints are augmented to the PI after being premuliplied by the *Lagrange multipliers* λ_k to obtain,

$$\Im = h(x_N, N) + \sum_{k=i}^{N-1} J^k(x_k, u_k) + \lambda_{k+1}^T [f^k(x_k, u_k) - x_{k+1}] \tag{1.16}$$

The *Hamiltonian* function is then given as [22],

$$H^k(x_k, u_k) = J^k(x_k, u_k) + \lambda_{k+1}^T f^k \tag{1.16}$$

Then, the necessary conditions for the minimization of (1.16) are,

$$x_{k+1} = \frac{\partial H^k}{\partial \lambda_{k+1}} = f^k(x_k, u_k) \tag{1.17}$$

$$\lambda_k = \frac{\partial H^k}{\partial x_k} = \left(\frac{\partial f^k}{\partial x_k}\right)^T \lambda_{k+1} + \frac{\partial J^k}{\partial x_k} \tag{1.18}$$

$$0 = \frac{\partial H^k}{\partial u_k} = \left(\frac{\partial f^k}{\partial u_k}\right)^T \lambda_{k+1} + \frac{\partial J^k}{\partial u_k} \tag{1.19}$$

with the boundary conditions,

$$\left[\frac{\partial J^i}{\partial x_i} + \left(\frac{\partial f^i}{\partial x_i}\right)^T \lambda_{i-1}\right]^T dx_i = 0 \tag{1.20}$$

and

$$\left[\frac{\partial h}{\partial x_N} - \lambda_N\right]^T dx_N = 0 \tag{1.21}$$

Equations (1.18) and (1.19) are known as the adjoint and costate equations respectively. Equation (1.20) is true for the case of unbounded control input.

The discrete optimal control problem ends up with a two-point boundary value problem, in terms of a difference equation with initial and final conditions to be satisfied. The solution to this type of problems is prohibitively complex specially when the input-output dimension is high and/or the degree of nonlinearity of the system is high.

A variety of analytical and numerical techniques have been developed to solve these types of problems. The techniques result in two main classes of solutions, namely open loop solution and closed loop solution. The following two paragraphs describe them in more details.

Open Loop Solution: Different numerical techniques have been developed to solve the optimal control problem. Examples of such techniques are dynamic programming, gradient algorithm, quasilinearization, and conjugate gradient algorithms [10]. These techniques produce an "open loop" solution for the problem. The solution is obtained off-line and then applied to the system during the real-time implementation without feeding back any of the system states. This solution is suitable only for the specified initial states and system parameters. Any disturbance to the system parameters or change in the initial states, while the open loop control signal is applied to the system, might lead to unacceptable results. This is why these techniques are rarely used in practical implementation of control systems. Instead, they are mostly used for simulation purposes to evaluate new solution techniques and study of systems behavior towards different control strategies.

Closed Loop Solution: It is always desirable to have the controller in closed loop form. This gives the control system more robustness to uncertainties in parameters and environment. In the closed loop control, the control signal is a function of the actual system states and not the predicted states as in the open loop solution. Few techniques in the literature have been successfully used to obtain the closed loop solution of the optimal controller. The Extremal Field method [10] is one of these techniques where the state space is filled with extremal optimal paths. During implementation, the optimal path is chosen depending on the current system states. The disadvantage of this techniques is the huge amount of storage space needed to store all optimal trajectories.

Other techniques also have been developed to obtain closed-form approximation of the optimal control solution. The singular perturbation technique (SPT) is one

of the most successful techniques. It produces closed-form zeroth and higher order approximation to the optimal controller. This technique was used to solve different optimal control problems [11, 40, 44]. However, coming up with the analytical solution of the optimal controller using this technique is not always an easy task. It also involves some decisions about the system that need to be made such as the time scale separation between the system states. This information is not always available to the designer [41].

Chapter II

APPLICATION OF ANN TO SOLVE OPTIMAL CONTROL PROBLEMS:

A LITERATURE REVIEW

There have been considerable research interest in utilizing neural networks into control applications. Artificial neural networks have some suitable features that motivated this interest. These features may be summarized as follows [32].

1. They are capable of learning any function provided that adequate information is provided during the training.
2. They have the ability of nonlinear mapping which makes them suitable for solving highly nonlinear control problems.
3. They require less information about the plant to be controlled and thus can control under wider range of uncertainty.
4. Because of their massive parallelism, ANN's entertain very fast multiprocessing.
5. Effect of partial damage of the neuro-controllers is smaller than that of the classical controllers due to their massive parallel structure.

In the literature, there exists a number of structures for neuro-controllers. These structures are based on well-established and well analyzed control principles. A brief discussion about these structures is provided in the following.

*1. **Supervised control.*** In this structure a neural controller mimics the human action in situation where the design of a classical automatic controller is impossible [46]. This type of control action usually can not be described by an analytical model. To train the network, the sensory information normally used by the human operator may be provided as inputs to the network and the network may be trained using a supervised method.

*2. **Direct Inverse Control.*** In this approach, the controller directly assumes the inverse model of the plant. The controller is cascaded with the plant to result in identity mapping between the reference signal and the output of the plant. The lack of feedback in this approach causes the lack of its robustness. Also, this approach can not be applied to non-minimum phase plants. This approach has been used in [25].

*3. **Model Reference Control.*** In this approach, the desired response of the plant is specified through the response of a suitable stable reference model. The error between the output of the reference model and the plant output is used for updating the parameters of the neuro-controller. A sizable amount of research work exists in this area. Narendra and Parthasarathy [28] described some frame works for obtaining direct and indirect model reference adaptive controllers using neural networks. In addition, Ahmed [3, 4] implemented neural networks to solve direct model reference adaptive control problems.

*4. **Internal Model Control (IMC).*** In this structure, a neural net-based plant model is used in parallel to the plant. The error between the plant output and the model outputs is fed back to the controller. The controller consists of two parts. The first part is an inverse neuro-model controller and the other is a subsystem, usually a

linear filter, which can be designed to introduce desirable robustness and tracking response to the closed-loop system. The only limitation to this approach is that it can only be applied to plants with stable inverse. This scheme was discussed and examined by Morari and Zafiriou [26] and Hunt *et al.* [15].

5. ***Optimal and Predictive Control.*** In this approach, a neural net-model is used to provide prediction for the future plant response over a specified horizon. These predictions are passed to an optimization routine which in turn attempts to minimize a specific performance index by suitably adjusting the parameters of the neuro-controller. The controller is placed in a feedback loop with the plant resulting in desirable stability properties for nonlinear systems [15]. The paper by Keerthi and Gilbert [18] is considered as one of the first efforts in investigating this type of control structures. Since this approach is directly related to our work, in the following we discuss this approach in details.

ANN's have been applied efficiently to solve optimal control problems that were considered otherwise difficult. In the literature, the optimal neuro-control problem has been solved using three distinctive approaches.

A number of researchers considered the trajectory learning scheme. In this scheme, the open loop control is first computed through solution of the boundary value problem using a standard numerical techniques. Then an ANN is trained to approximate a closed-loop relationship between the system states and the precomputed input. The employment of such a closed loop neuro-controller is expected to be more robust compared to the open loop control. Some of the papers that reported this technique are those by McDermott and Athans [24], Zakrzewski and Mohler [50], and Baldi [8].

The second approach in optimal neuro-control is based on directly obtaining the closed loop neural controller through optimization. The optimization usually requires gradient computation considering the complete closed loop dynamics. Most of the literature reporting optimal neuro-control are based on this approach. For example, a numerical approximation law of time-optimal control by means of neural networks has been proposed by Niesler and Plessis [31]. The advantage of this techniques is that no prior assumptions are made on the plant. Also, the technique is flexible to inclusion of problem-specific constraints. In this approach, first an initial number of time steps is chosen. Then, the ANN is trained to force the final states to some desired values. After each update of the network weights, the final states error is compared to a small number. If it is smaller, the number of steps is decremented by one and the training is resumed. When this doesn't occur within specific number of iterations then the number of steps is incremented by one and the training is terminated.

The algorithms used to compute gradients in neural systems can be categorized into three general classes. They are Dynamic Backpropagation (DB), Backpropagation Through Time (BPTT), and Block Partial Derivative (BPD). These algorithms extend the classical BP algorithm to dynamic networks. The DB algorithm is proposed by Narendra and Parthasarathy [27, 28] while the BPTT algorithm was proposed by Werbos [45] and applied by Plumer [38] and Park [37]. The BPD concept was proposed by Ahmed [3] and applied to solve model reference direct neuro-adaptive control. BPD has the following advantages over the DB and BPTT.

1. BPD is extremely simple to apply. It does not follow the cumbersome approach of DB and BPTT in employing the chain rule of derivative computation. Rather,

it simplifies the derivation using graph (or Block-diagram) reduction technique that is well understood by control engineers.

2. BPD computation progresses in hierarchy. Once the BPD for a block is obtained, the BPD computation of the outer block can make use of it. This makes the BPD approach attractive for complex networks. This feature can be used in BPD derivation as well as in BPD computation. DB and BPTT do not enjoy this feature.
3. The BPD approach is applicable to networks with arbitrary configuration. Its applicability in controller training is limited only by the imagination of the control designer.

The BPD concept has not yet been applied in solving optimal control problems. The developments in this thesis are based on this approach where the BPD concept is utilized to compute the gradient of the closed loop dynamics to implement direct state-feedback optimal control.

The gradient based algorithms usually have two limitations, slow convergence and convergence to a local minimum. However, two other algorithms have been found in the literature known as the Quick Prop and R Prop. The Quick Prop uses a second order equation related to Newton's method for weight update [42]. The R Prop updates the network weights according to the sign of the gradient [40, 31]. It has been reported that both of these methods yield comparably fast convergence.

The third approach of solving optimal control problems using neural networks is based on converting the optimal control problem into boundary value problems (BVP). The ANN is then trained to satisfy the equations defined by the BVP. One of the papers that incorporate this technique is by Biswas [9] where he developed a

Hopfied[1] Neural Network based scheme to solve a class of optimal control problems. The problem is first transformed to a two-point boundary value problem which is then converted to minimization of an error function. The error function is mapped into the energy function of the Hopfield-Tank Neural Network. Another related study is due to Li and Chen [23] where they developed a neural-type network to compute minimum energy path in real time under two-point boundary conditions. The two-point BVP is first converted into a special domain suitable for analog computing. Then, a resistive network is built to implement the new formulation utilizing elementary linear circuit laws. Further work on this approach can be found in Plumer [38] and Balakrishnan and Biega [7].

While most of the researchers considered deterministic plants, there exist some effort in dealing with stochastic systems [35, 30, 36, 12]. Concerning the performance index, most of the work reviewed is based on general indexes. However, in developments such as [37], only quadratic performance index is considered.

Several researches also reported optimal neuro-control for specific applications. Balakrishnan and Biega [7] developed a neural net-based controller to improve the performance of a homing missile guidance. Other study by Yen *et al.* [49] used neural networks to implement near-time optimal position control for DC motor servosystems. The training procedure was based on trajectory learning of an experimentally generated near-time-optimal solutions. Neural networks were also used in developing optimal control policies for flexible pointing structures. Examples of studies in this field include those by Adeli and Park [1] and Yen [48].

---

[1] Hofield networks consist of an input layer and a single layer known as the Hopfield layer. Each output from the neurons in the Hopfield layer is weighted and fed back to the inputs of all other neurons in the layer. Hopfield NNs are mostly used for binary data representation [33].

Finally, a paper by Timofeev and Bogdanov [43] reported application of neural networks to synthesize optimal controllers for robot systems.

CHAPTER III

NEURAL NET STATE FEEDBACK OPTIMAL CONTROL

3.1 Overview of the Proposed Scheme

Consider the nonlinear plant described by the following state variable model,

$$x(k+1) = f(x(k), u(k), k) \qquad (3.1)$$

where $x(k) \in \mathbf{R}^n$ is the state vector and $u(k) \in \mathbf{R}^m$ is the control vector. $f(.)$ is an n-valued known function, although no specific structure is assumed for it. The control $u(k)$ is generated by a Multi-layered Feed Forward Neural Network (MFNN) as shown in Figure 3.1 forming a closed loop state feedback tracking control. This state feedback strategy imitates the optimal quadratic control of linear systems. The state feedback makes the control system robust while the normalized time τ as an input to the neural controller allows the controller to be time varying, mimicking the time varying state feedback gain of its linear counter part. The desired states vector $x_d(k+1)$ is also fed to the neural network controller as an additional input.

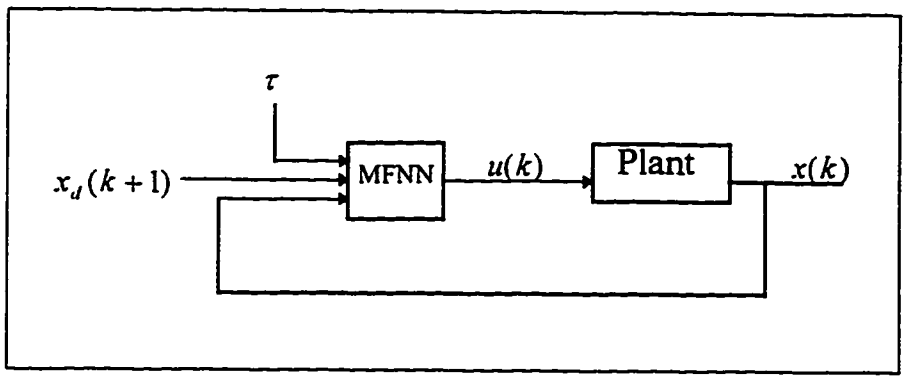


Figure 3.1 General structure of the state feedback neural net based optimal control.

The objective of the control design is to train the neural network such that the following general performance index (PI) is minimized,

$$\Im = \sum_{k=1}^{N} J(\varepsilon(k), u(k-1), k) \qquad (3.2)$$

where $\varepsilon(k)$ is the error signal between the desired and actual states, i.e.,

$$\varepsilon(k) = x_d(k) - x(k). \qquad (3.3)$$

Figure 3.2 shows how the training process is carried out. One way to minimize the PI in equation (3.2) is to train the network based on the steepest decent algorithm where the weights of the network are updated along the direction negative to the gradient. i.e.,

$$W_{i+1} = W_i - \eta_i \left(\frac{\partial^D \Im}{\partial W_i} \right)^T \qquad (3.4)$$

The $n_W \times 1$ vector W contains all the weights of the MFNN. The index i indicates the iteration number and η_i is the '*adaptation rate*'. It is important to notice that the gradient is computed with respect to block D that contains the entire closed loop system. The following section considers the computation of this gradient utilizing the BPD concept.

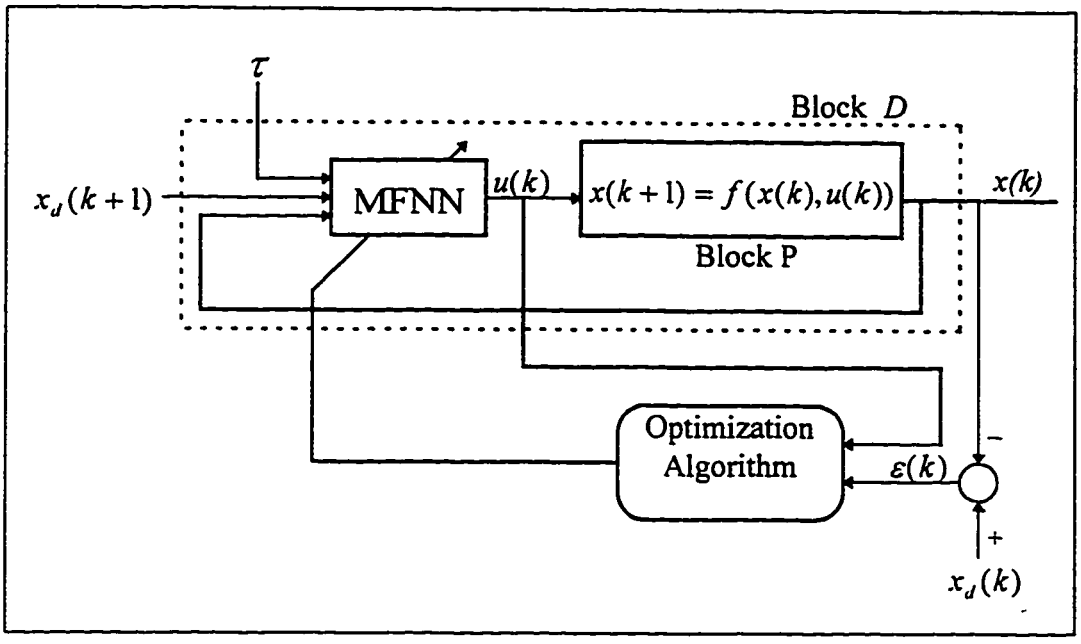


Figure 3.2 The MFNN controller training scheme.

3.2 Computation of the Gradient

The chain rule may be used to obtain the gradient in terms of the block partial derivatives of the error and input signals as follows,

$$\frac{\partial^D \Im}{\partial W_i} = \sum_{k=1}^{N} \left[\frac{\partial J(k)}{\partial \varepsilon(k)} \frac{\partial^D \varepsilon(k)}{\partial W_i} + \frac{\partial J(k)}{\partial u(k-1)} \frac{\partial^D u(k-1)}{\partial W_i} \right]. \tag{3.5}$$

The partial derivatives $\frac{\partial J(k)}{\partial \varepsilon(k)}$ and $\frac{\partial J(k)}{\partial u(k-1)}$ can be computed explicitly from equation (3.2). Furthermore, from equation (3.3) we can write,

$$\frac{\partial^D \varepsilon(k)}{\partial W_i} = -\frac{\partial^D x(k)}{\partial W_i} \tag{3.6}$$

since the change in the weights has no effect on the desired states $x_d(k)$.

Computation of the Sensitivity Derivatives: Our aim now is to compute the two sensitivity derivatives $\frac{\partial^D x(k)}{\partial W_i}$ and $\frac{\partial^D u(k)}{\partial W_i}$ appearing in equations 3.5 and 3.6. This can be achieved by the utilization of the BPD concept. First, the BPD's for each block inside *D* (see Figure 3.2) are computed following the procedure described in 1.4.5. In this case, the weights are set as auxiliary input variables to the MFNN controller. The resulting linearized diagram is shown in Figure 3.3. The superscript (+) indicates the ordered partial derivatives which are computed from the Backpropagation algorithm [45]. The transfer function $\Phi(q)$ for the linearized plant is obtained from Figure 3.3 as,

$$\Phi(q) = [qI - A(k)]^{-1} B(k) \qquad \text{(dim: } n\times m\text{)} \qquad (3.7)$$

where q is a time delay parameter, and the matrices *A(k)* (dim: $n\times n$) and *B(k)* (dim: $n\times m$) denote the plant Jacobians $\frac{\partial^P x(k+1)}{\partial x(k)}$ and $\frac{\partial^P x(k+1)}{\partial u(k)}$. Their expressions are derived directly from equation 3.1 as,

$$A(k) \equiv \frac{\partial^P x(k+1)}{\partial x(k)} = \frac{\partial f(x(k),u(k))}{\partial x(k)} \qquad \text{(dim: } n\times n\text{)} \quad (3.8)$$

$$B(k) \equiv \frac{\partial^P x(k+1)}{\partial u(k)} = \frac{\partial f(x(k),u(k))}{\partial u(k)}. \qquad \text{(dim: } n\times m\text{)} \quad (3.9)$$

Examining Figure 3.3, we also find that,

$$\partial u(k) = \frac{\partial^+ u(k)}{\partial W_i}\partial W_i + \frac{\partial^+ u(k)}{\partial x(k)}\partial x(k) + \frac{\partial^+ u(k)}{\partial \tau}\partial \tau + \frac{\partial^+ u(k)}{\partial x_d(k+1)}\partial x_d(k+1). \qquad (3.10a)$$

and

$$\partial x(k) = \Phi(q)\partial u(k)\,. \qquad (3.10b)$$

These two equations characterize the linearized relationship inside the D block. Dividing both equation 3.10a and 3.10b by the elements of ∂W_i concatenating and realizing that $\frac{\partial \tau}{\partial W_i} = \frac{\partial x_d(k+1)}{\partial W_i} = 0$ we have,

$$\frac{\partial^D u(k)}{\partial W_i} = \frac{\partial^+ u(k)}{\partial W_i} + \frac{\partial^+ u(k)}{\partial x(k)}\frac{\partial^D x(k)}{\partial W_i}, \qquad \text{(dim: } m\times n_W\text{)} \qquad (3.11a)$$

and

$$\frac{\partial^D x(k)}{\partial W_i} = \Phi(q)\frac{\partial^D u(k)}{\partial W_i}. \qquad \text{(dim: } n\times n_W\text{)} \qquad (3.11b)$$

Substituting back equation 3.11b into 3.11a and taking $\frac{\partial^D u(k)}{\partial W_i}$ as common factor we may get,

$$\frac{\partial^D u(k)}{\partial W_i} = \left[I - \frac{\partial^+ u(k)}{\partial x(k)}\Phi(q) \right]^{-1} \frac{\partial^+ u(k)}{\partial W_i}. \qquad \text{(dim: } m\times n_W\text{)} \qquad (3.12)$$

Thus the gradient in equation 3.5 can be computed employing equations (3.6, 3.7, 3.8, 3.9, 3.11b, and 3.12).

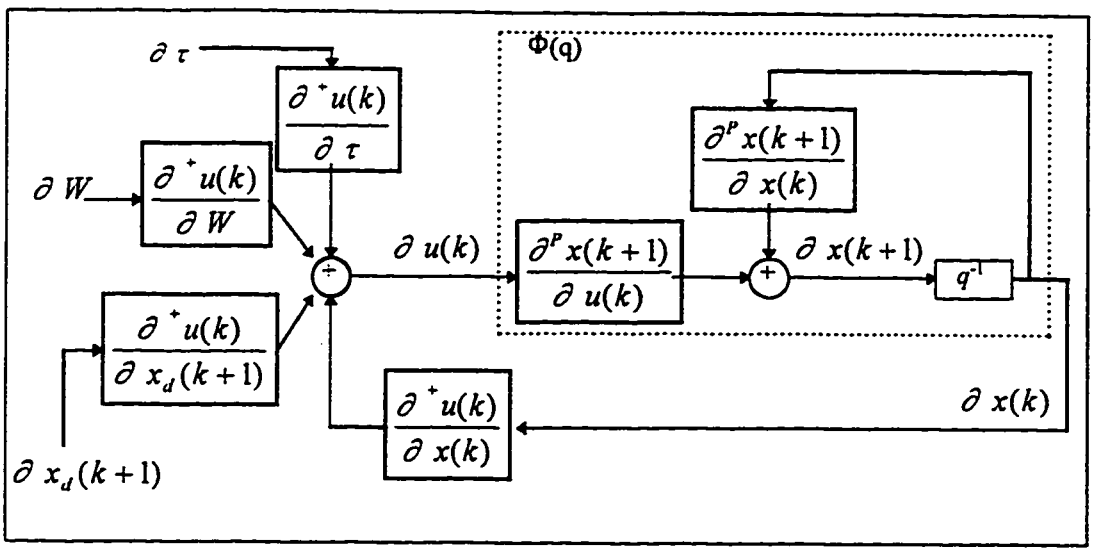


Figure 3.3 The linearized diagram for BPD computation.

3.2.1 Quadratic Performance Index

Use of quadratic performance index is common in optimal control problems. The designer has the flexibility of choosing the weighting matrices to achieve suitable tracking and at the same time impose a penalty on the control effort applied. In this section, a computational scheme for the gradient will be derived that is applicable to the quadratic performance indexes.

Consider the quadratic performance index,

$$\Im \equiv \sum_{k=1}^{N} J(k) = \tfrac{1}{2} \sum_{k=1}^{N} \left\{ \|\varepsilon(k)\|^2_{Q(k)} + \|u(k-1)\|^2_{R(k-1)} \right\}. \qquad (3.13)$$

where $Q(k)$ and $R(k)$ are, respectively, positive semi-definite and positive definite real symmetric weighting matrices of appropriate dimension and $\|\cdot\|$ denotes the

euclidean norm. For this specific PI,

$$\frac{\partial J(k)}{\partial \varepsilon(k)} = \varepsilon^T(k)Q(k) \qquad \text{(dim: } 1\times n\text{)} \qquad (3.14)$$

$$\frac{\partial J(k)}{\partial u(k-1)} = u^T(k-1)R(k-1) \qquad \text{(dim: } 1\times m\text{)} \qquad (3.15)$$

Now using equation (3.5), the gradient can be written as,

$$\frac{\partial^D \Im}{\partial W_i} = \sum_{k=1}^{N}\left[-\varepsilon^T(k)Q(k)\Psi(k) + u^T(k-1)R(k-1)\Pi(k-1)\right], \qquad (3.16)$$

with

$$\Psi(k) \equiv \Psi(W_i,k) = \frac{\partial^D x(k)}{\partial W_i} \qquad \text{(dim: } n\times n_W\text{)} \qquad (3.17)$$

$$\Pi(k) \equiv \Pi(W_i,k) = \frac{\partial^D u(k)}{\partial W_i}. \qquad \text{(dim: } m\times n_W\text{)} \qquad (3.18)$$

The sensitivity derivatives $\Psi(k)$ and $\Pi(k)$ can be computed using equations (3.11b) and (3.12). Finally, the weights of the MFNN controller can be updated using equation (3.4) which takes the following form,

$$W_{i+1} = W_i - \eta_i \sum_{k=1}^{N}\left[-\varepsilon^T(k)Q(k)\Psi(k) + u^T(k-1)R(k-1)\Pi(k)\right]^T \qquad (3.19)$$

3.2.2 Adaptation Rate:

The adaptation rate plays an important rule in the training of the MFNN. A small value makes the training very slow while large value causes weight oscillation and

may cause the training to diverge. The adaptation rate η could be scalar, matrix, constant, or time varying.

The use of constant adaptation rate is found attractive in many cases [24, 38]. Using adequately small rate, the convergence can be made smooth. Thus many conclusions about the training process can be drawn at early stages of the training such as divergence and speed of convergence. Since the training is conducted off-line, the choice of the constant learning rate may be made using trial and error bases. Usually, it doesn't take a long training time to discover the goodness of this choice.

Some algorithms for time varying adaptation rate have been developed [2, 12] to speed up and/or guarantee convergence. One of these algorithms is developed in [2]. It was shown that this algorithm ensures fast convergence and at the same time ensures local stability in the adaptive control system. The algorithm is based on the error function linearization in the weight space. The algorithm suggests that choosing the following form for η ensures that the weights will always get closer to the target weights of the network.

$$\eta(i) = \frac{\mu(i)}{\bar{\epsilon} + \sum_{k=1}^{N} \left\| \Psi(k)\bar{Q}(k) \right\|^2 + \sum_{k=1}^{N} \left\| \Pi(k)\bar{R}(k-1) \right\|^2}, \tag{3.20}$$

with

$$0 \le \mu(i) \le 2;$$

$$\bar{\epsilon} > 0;$$

$$Q(k) = \bar{Q}(k)\bar{Q}^T(k);$$

$$R(k) = \bar{R}(k)\bar{R}^T(k).$$

The small positive parameter $\bar{\epsilon}$ is added to avoid division by zero. The variable $\mu(i)$ is allowed to vary linearly according to,

$$\mu(i+1) = \mu_f - \mu_r\left[\mu_f - \mu(i)\right]; \quad \mu(0) = \mu_0 \tag{3.21}$$

where μ_0, μ_f and μ_r are respectively the initial value, final value and rate of change of μ. According to [2], a suitable choice of μ_r is between 0.95 and 1.

3.3 Application of the Algorithm to Typical Optimal Control Problems

The algorithm developed in the previous sections is an optimal tracking problem. However, the same algorithm can be used to solve other typical optimal control problems by proper choice of the PI parameters. This section demonstrates how the algorithm can be applied to solve regulation, terminal control, minimum control effort, time optimal control, and output tracking problems.

3.3.1 Regulator Problem

Regulator problem is a special class of the tracking problem with the reference signal is constant over a long period of time [20]. In this case, the reference signal is referred to as the set point. The main objective of the regulator problem usually is to maintain the control variables near the set point in spite of disturbance. To apply the previous results to regulator problems, we simply set $x_d(k)$ to a constant value for all time. If $x_d(k)$ is a nonzero set point then the steady state

error may not be zero. This situation can be corrected by shifting the origin of the state space to the set point [20].

3.3.2 Terminal Control Problem

Terminal control problem is usually concerned with the minimization of the deviation of the system's final states from their desired values [19]. A performance index for the terminal control problem could be,

$$\Im = [x(N) - x_d(N)]^T \mathbf{H}[x(N) - x_d(N)]. \qquad (3.22)$$

where **H** is real symmetric positive semi-definite weighting matrix of appropriate dimension. The elements of the matrix **H** are chosen to weight the relative importance of the deviation of each controlled variable from its desired value.

The developed MFNN-control algorithm can be adapted easily to solve the terminal control problem. To do so, the lower sum index in the quadratic performance index, equation (3.13), is set to N and $Q(N)$ is set to **H**.

3.3.3 Minimum Control Effort Problem

The minimum control effort problem is concerned with transferring a system from an initial state $x(0)$ to a specified target set S, with minimum expenditure of

control effort [19]. One typical performance index is given by,

$$\Im = \sum_{k=1}^{N} \|u(k)\|^2_{R(k)} \tag{3.23}$$

and the control problem is referred to as the *minimum energy problem.* An example of minimum energy problem is the minimization of source energy dissipation in an electric circuit. Minimum energy problem is a special case of the general quadratic performance index and thus equation 3.19 applies with $Q(k)$ is set to zero for all k.

Another typical minimum control effort problem has the performance index,

$$\Im = \sum_{k=1}^{N} |u(k)|_{R(k)} \tag{3.24}$$

This problem is referred to as the *minimum fuel problem.* Example of minimum fuel problems includes sending a missile to a specific final coordinate with minimum thrust. Since the PI involves an absolute term, the gradient need to be modified. For example, the gradient for the single input case is given by,

$$\frac{\partial^D \Im}{\partial W_i} = \sum_{k=1}^{N} \left[sign(u(k)) \frac{\partial^D u(k)}{\partial W_i} \right]. \tag{3.25}$$

where the BPD $\frac{\partial^D u(k)}{\partial W_i}$ is defined in equation 3.12.

The control effort could be either unconstrained or constrained by upper and lower limits. For the unconstrained case, no modifications are necessary. For the

constrained case, the following statement is added to the problem,

$$u_{\min} \leq u(k) \leq u_{\max} \tag{3.26}$$

This constraint may easily be accommodated in the algorithm. Whenever the computed input falls outside the range, it is truncated accordingly. Furthermore, whenever the truncation takes place, all partial derivatives of the input signal with respect to other variables are set to zero.

3.3.4 Time Optimal Control Problem

The minimum time problem is concerned with transferring a system from an arbitrary initial state x_0 to a specific target set **S** in minimum time [19]. An approach based on the bisection algorithm is developed here to solve the minimum time problem. The basic idea is to start with an arbitrarily large final time and then train the MFNN to drive the system to the desired final states within this time. If this is achieved within certain number of iterations, then the final time is decremented, otherwise, it is incremented. The MFNN training is then restarted. In this approach, the number of sampling points N is fixed and the sampling time T_S is adapted until the optimal sampling time T_S^* is obtained resulting in an optimal final time $t_f^*=NT_S^*$.

The problem can be divided into two main tasks. The first task is the adaptation of the final time by changing T_S. The second task is a terminal control problem where the MFNN is trained to derive the system to the desired final states within the adapted final time. The developed approach is illustrated by Figure 3.4 and can be summarized in the following steps.

Step 1. Choose initial "good" and "bad" sampling times. A good sampling time γ is the time that enables the MFNN to derive the final states of the system to their desired values within certain number of iterations and a bad sampling time β is the one that fails to do that. Initial guess for γ and β could be any values larger and smaller than T_S^* respectively. Next, set the iteration index i to zero and the sampling time to γ and go to step 2 to start the first *trial*.

Step 2. Increment the iterations index i by one and conduct one batch iteration to minimize the PI,

$$\Im = [x(N) - x_d(N)]^T \mathbf{H}[x(N) - x_d(N)],$$

using the updated sampling time.

Step 3. Compute the value of the PI, which is a measure of how close the final states are from their desired values. If the computed value of the PI is less than a threshold δ, the sampling time is considered good. In this case, assign the sampling time to γ and skip to step 5. If the value of the PI is still grater than δ, then go to step 4.

Step 4. Check whether one of the following measures is satisfied,

- the training process reaches steady state. One way to check that is by examining the rate of change of the performance index $\Delta\Im$. If it is less than a small percentage (0.1% is used in our study) then we can assume that the training has reached steady state.
- the number of iterations i exceeds a maximum allowed number i_{max}.

If any of these two measures is satisfied, then the sampling time is considered bad. In this case, assign the sampling time to β and go to step 5. On the other hand, if neither of the two measures are satisfied then go back to step 2. For the situations where the steady state can not be determined, the first measure can be ignored.

Step 5. Reset the iterations index to zero and update the sampling time for the next trial. Choose the new value of the sampling time to be in the middle between the latest good and bad sampling times as illustrated in Figure 3.5. Proceed to step 6.

Step 6. Repeat the steps from 2 to 5 until acceptable accuracy of T_S is obtained. A possible termination criteria could be,

$\Delta T_S \equiv T_S - \gamma \leq \rho; \qquad \rho > 0.$

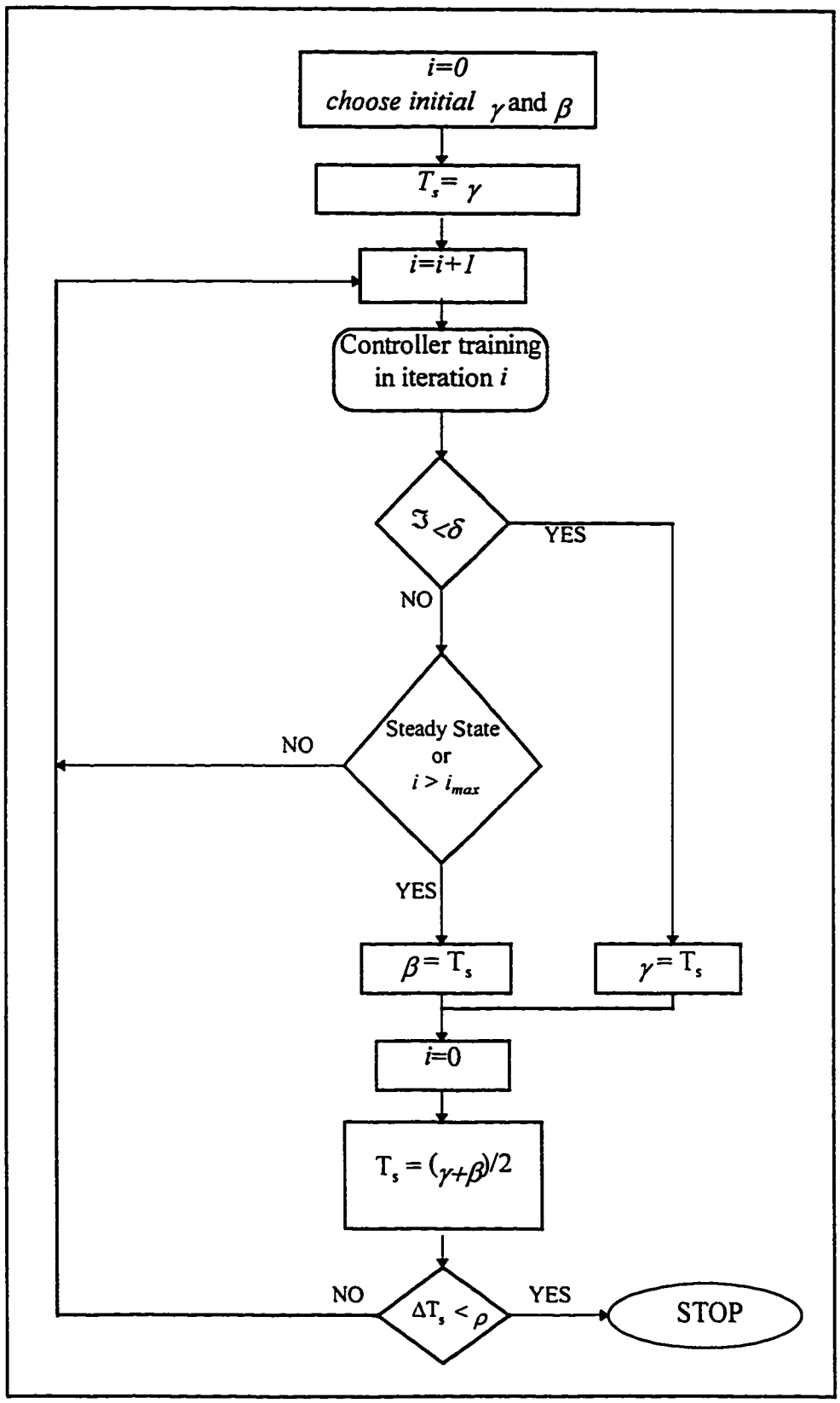


Figure 3.4 Flow chart of the time-optimal control algorithm.

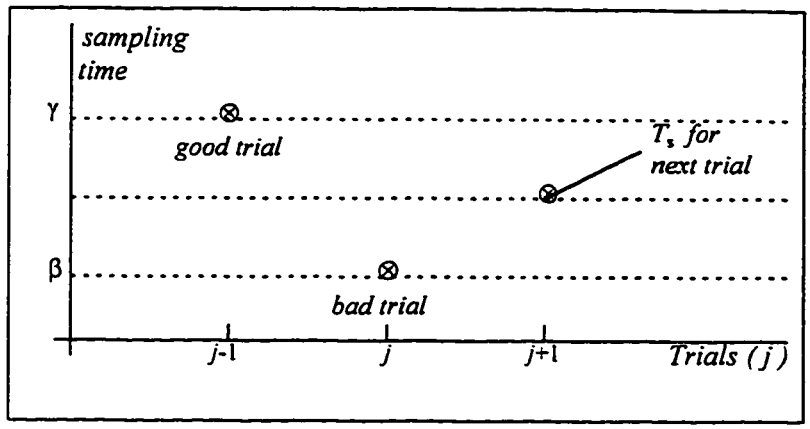


Figure 3.5 Sampling time for the next trial of the time-optimal control problem.

Programming Considerations: A certain sampling time may cause the problem to diverge and thus the whole training process may diverge. To overcome this problem, we need first to add a divergence measure to the three measures listed above. In Matlab. the divergence could be indicated by one of the variables, such as the control signal, having the **NaN** value. If divergence took place, then the first iteration after updating T_S should start with the weights which correspond to the latest good sampling time. This requires that whenever γ is updated, the weights of the MFNN should be stored. This treatment will guarantee that the training will not start from weight corresponding to a divergent training trial.

3.3.5 Servo Problem with Integrator

This is similar to the tracking problem as mentioned above, however is relatively rugged against modeling errors and is applicable when the desired output (or state) disturbance has only low frequency components. This scheme is also capable of rejecting the low frequency disturbances entering into the system. This concept is based on the classical control design where the error in the steady-state output is eliminated by introducing integrators. The introduction of integrators causes high loop gain at low frequencies causing the control system robust against low frequency modeling error and low frequency disturbances [6]. This scheme is generally applied to time-invariant plants.

A typical neural net based servo system with integrators is shown in Figure 3.6, where a square plant (i.e. having same number of input and output) is considered for output tracking. The plant output is represented as,

$$y(k)=h(x(k)) \tag{3.27}$$

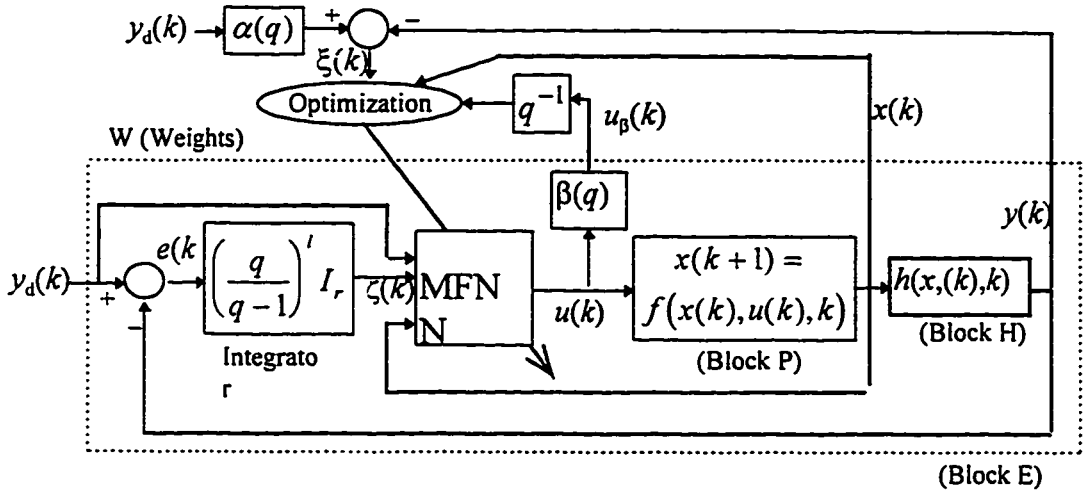
where $y(k)\in\mathbf{R}^r$ and $h(.)$ is an r-valued known function. The integrator has been represented using the advance operator q. l is the order of the integrator which should be chosen based on the normal operating setpoint to the plant. For example for a step-like setpoint, it should be taken as 1. For a ramp-like setpoint it should be chosen as 2 etc. The error feedback through the integrator ensures steady state tracking, while the direct injection of the setpoint to the controller causes the control system to quickly respond to a change in the reference input. In the figure, $y_d(k)$ represents the desired output. The control design, in this case, requires

adjusting the weights of the neural controller such that the closed loop system remains stable and at the same time the transients are satisfactory.

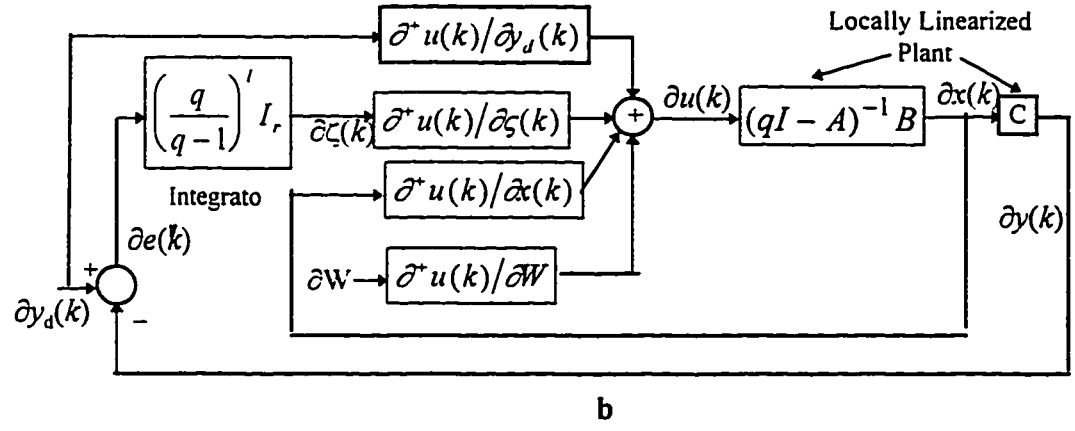
A general performance index for the control design in this case may be expressed as,

$$\Im = \tfrac{1}{2}\sum_{k=1}^{N}\left[\left\|\alpha(q)y_d(k) - y(k)\right\|^2_{M(k)} + \left\|x(k)\right\|^2_{Q(k)} + \left\|\beta(q)u(k-1)\right\|^2_{R(k)}\right] \qquad (3.28)$$

where $\alpha(q)$ and $\beta(q)$ are linear filters which are required to be suitably chosen. The choice of $\alpha(q)$ should ensure the steady state tracking of the desired output. The choice of the filter $\beta(q)$ and the real semi-definite symmetric matrices $M(k)$, $Q(k)$ and $R(k)$ shape up the transient response of the closed loop system.



a



b

Figure 3. 6 Controller training for the output servo problem (with integrator) **a.** Training configuration **b.** Linearized diagram for gradient computation

Now consider the training scheme shown in Figure 3.6a and performance index is given by equation (3.28). As before, the weights of the MFNN are updated according to the following rule.

$$W_{k+1} = W_k - \eta_k \left(\frac{\partial^E \Im}{\partial W_k} \right)^T \tag{3.29}$$

where η_i as earlier represents the 'adaptation rate'. The gradient can be calculated from,

$$\frac{\partial^E \Im}{\partial W_k} = \sum_{k=1}^{N} \left[-\xi^T(k) M(k) \Upsilon(k) + x^T(k) Q(k) \Psi(k) + u_\beta^T(k-1) \Gamma(k-1) \right], \tag{3.30}$$

where

$$\Upsilon(k) \equiv \Upsilon(W_i, k) = \frac{\partial^E y(k)}{\partial W_i} \qquad (\text{dim: } r \times n_W)$$

$$\xi(k) \equiv \xi(W_i, k) = \alpha(q) y_d(k) - y(k) \qquad (\text{dim: } r \times 1)$$

$$u_\beta(k) = \beta(q) u(k)$$

$$\Gamma(k) \equiv \Gamma(W_i, k) = \beta(q) \Pi(k) \qquad (\text{dim: } m \times n_W)$$

The sensitivity derivatives $\Psi(k)$ and $\Pi(k)$ are defined in an analogous way as in equations (3.17) and (3.18) with the exception that they along with $\Upsilon(k)$ must be computed for the block E that contains the entire closed loop system as shown in Figure 3.6a, otherwise they will not represent the true descent.

***Computation of the Sensitivity derivatives*:** In order to compute the sensitivity derivatives, in Figure 3.6b we draw the linearized network for the block E. In Figure 3.6b, $\partial^+ u(k)/\partial W$ (dim: $m \times n_W$), $\partial^+ u(k)/\partial x(k)$ (dim: $m \times n$), $\partial^+ u(k)/\partial \zeta(k)$ (dim: $m \times r$) and $\partial^+ u(k)/\partial \hat{c}_d(k)$ (dim: $m \times r$), as earlier, denote the ordered partial

derivatives (OPD) of the MFNN, which can be computed using the conventional back propagation algorithm.

The matrices A (dim: $n\times n$), B (dim: $n\times m$) and C (dim: $r\times n$) in the figure belong to the locally linearized plant equation which are obtained in terms of the BPD's. A and B are defined in equations (3.8) and (3.9) while C is obtained as

$$C = \frac{\partial^H y(k)}{\partial x(k)} = \frac{\partial(h(x(k)))}{\partial x(k)} \tag{3.31}$$

where the block H contains the state-output relationship as shown in Figure 3.6**a**. These BPD's can be readily calculated from the known functions $f(.)$ and $h(.)$.

From the linearized network drawn in Figure 3.6**b**, one may get the following expressions for $\Pi(k)$ and $\Psi(k)$

$$\Upsilon(k) = C\Psi(k) \tag{3.32}$$

$$\Psi(k) = (qI - A)^{-1} B\Pi(k) \tag{3.33}$$

and

$$\Delta(q)\Pi(k) = \frac{\partial^+ u(k)}{\partial W_i} \tag{3.34}$$

where $$\Delta(q) = I_m + K(qI - A)^{-1} B + \left(\frac{q}{q-1}\right)^l SC(qI - A)^{-1} B$$

with $$S = \frac{\partial^+ u(k)}{\partial \varsigma(k)} \quad \text{(dim: } m\times r\text{)}$$

and $$K = -\frac{\partial^+ u(k)}{\partial x(k)} \quad \text{(dim: } m\times n\text{)}$$

Equations (3.30-3.34) provide everything necessary for the controller training in the output tracking problem with integrator. the adaptation rate η_i can be chosen by trial and error.

CHAPTER IV

APPLICATION OF THE MFNN OPTIMAL CONTROL TO SIMULATED PLANTS

A Matlab computer program has been developed to design the MFNN optimal controller. The program assumes the following.

- Number of states of the plant is unrestricted.
- The plant is single input.
- Performance index is quadratic.
- The MFNN has 2 hidden layers with unrestricted number of neurons.
- Learning rate is adaptive.

The program requires the following information to be specified by the user,

- the discrete dynamic equations of the plant,
- the plant Jacobians *A(k)* and *B(k)*,
- the number of time steps and step size of simulation,
- the learning rate adaptation parameters.
- performance index parameters.

The proposed algorithms have been applied to five different optimal control problems. In the first example, a linear plant is regulated with a quadratic PI. The second example demonstrates the application of the algorithm to a first order nonlinear plant with quadratic PI. In the third example, a nonquadratic performance index is considered to obtain the shortest distance between two points. The fourth example applies the algorithm to solve a typical process control problem known as the *stirred tank reactor problem.* The solutions obtained through our approach for the first four example problems are compared with the solutions obtained through the conventional methods. In the fifth problem, optimal tracking, regulation, terminal control , minimum control effort, minimum energy, minimum time, and output tracking problems are illustrated using a second order nonlinear plant.

4.1 Quadratic Regulation of a Linear Plant

To test the proposed algorithm, it is first applied to a linear system with quadratic PI. This problem is known as the Linear Quadratic Regulation (LQR) for which a theoretical solution is available in optimal control theory.

Problem Statement

Consider the PI

$$\mathfrak{I} = x(N)H(N)x(N) + \sum_{k=0}^{N-1}\left\{x^T(k)Q(k)x(k) + u^T(k)R(k)u(k)\right\}, \qquad (4.1)$$

where N is the number of time steps and the plant is described by the following linear dynamics

$$x(k+1) = A(k)x(k) + B(k)u(k). \qquad (4.2)$$

with $A(k)$ and $B(k)$ are chosen as,

$$A(k) = \begin{bmatrix} 1.1 & 0 \\ 0 & 1.2 \end{bmatrix} \quad ; \; B(k) = \begin{bmatrix} 1 \\ 2 \end{bmatrix}.$$

The weighting matrices are chosen to be,

$$Q(k)=0.01I_2 ; \quad R(k)=0.01; \quad H(k)=100.$$

where I_2 denotes a 2×2 identity matrix.

Problem Solution

Since the plant is linear, the Jacobians appear in equations (3.8) and (3.9) are simply the plant matrices $A(k)$ and $B(k)$. The Matlab program is executed with N=30 and a step size of 0.1 seconds.

Figure 4.1 shows the trajectory of the first state $x_1(k)$ for the first 400 iterations. Before starting the training and with the initial randomly-chosen weights, the trajectory is clearly unstable. As the training proceeds, the neural network controller forces the trajectory of $x_1(k)$ towards the optimal one. The trajectory corresponding to the next iterations are shown separately in Figure 4.2 in a magnified scale. The final trajectories of the system states and control effort have been compared to the exact ones obtained through solving the discrete Riccati equation [22] and the results are shown in Figure 4.3. The values of the PI obtained by the two techniques are listed in Table 4.1. The initial value of the PI before training has been 2×10^9.

Technique	**Performance Index**
MFNN State-Feedback Control	1.6764
Solution of the Riccati Equation	1.6663

Table 4.1 PI Values of the LQR problem.

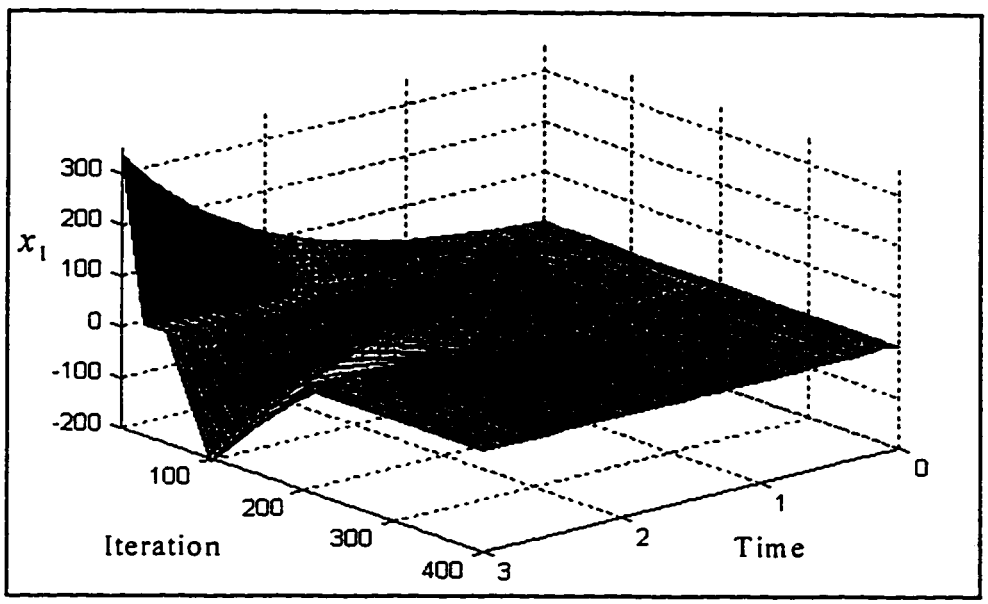


Figure 4.1 Trajectory of x_1 during training.

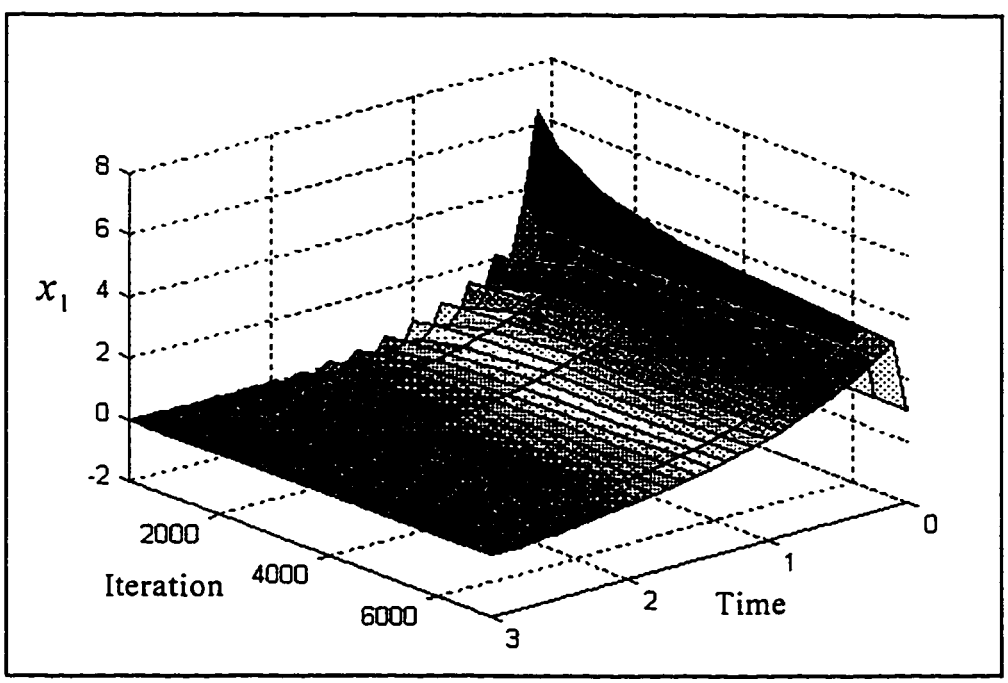


Figure 4.2 Continuation of the training process starting from iteration 400 up to 7000.

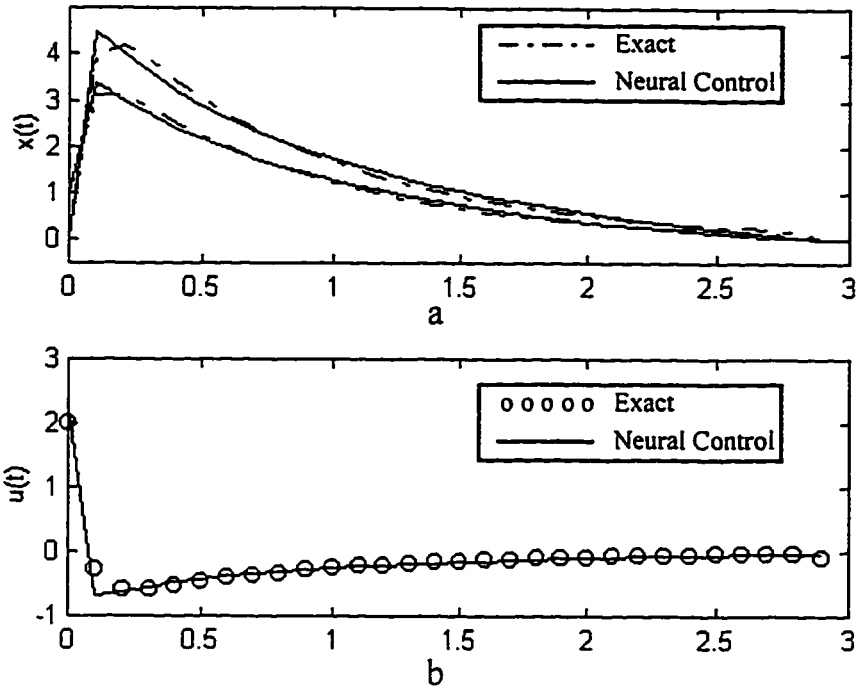


Figure 4.3 Final trajectories of the linear quadratic regulation problem.
a. system states. b. control effort.

4.2 A First Order Nonlinear Plant

Consider the first order continuous nonlinear plant described by

$$\dot{x}(t) = -x^2(t) + u(t), \qquad x(0)=10.0 \tag{4.3}$$

and the performance index given by

$$\Im(t) = \frac{1}{2}\int_0^{1.0}\left(x^2(t) + u^2(t)\right) dt\,. \tag{4.4}$$

To solve this problem we need first to discretize the dynamic equation and PI.

Using the forward difference we have

$$x(k+1) = x(k) + T_s(-x^2(k) + u(k)) \tag{4.5}$$

and

$$\Im(k) = \frac{1}{2}\sum_{k=0}^{N} T_s\left[x^2(k) + u^2(k)\right]. \tag{4.6}$$

Where N is the number of time steps and T_S is the step size. The Jacobians of the plant are evaluated from equations (3.8) and (3.9) as

$A(k) = 1\text{-}2T_S x(k)$ and $B(k) = T_S$

Further, the weighting matrices $Q(k)$ and $R(k)$ are

$Q(k) = T_S$

$R(k) = T_S$.

The developed Matlab program has been used to obtain the solution for this problem. The MFNN used have two hidden layers with five and three neurons respectively. The values of N and T_S are set to 61 and 0.0164 respectively. The problem converged after few number of iterations. The value of the PI as obtained through the proposed procedure is shown in Table 4.2 together with that obtained by quasilinearization. The optimal trajectory of the state and control effort are shown in Figure 4.4 while the decay of the PI is shown in Figure 4.5.

Technique	**Performance Index**
MFNN State-Feedback Control	4.5106
Quasilinearization	4.5109

Table 4.2 Comparison between the PI values for Example 4.2.

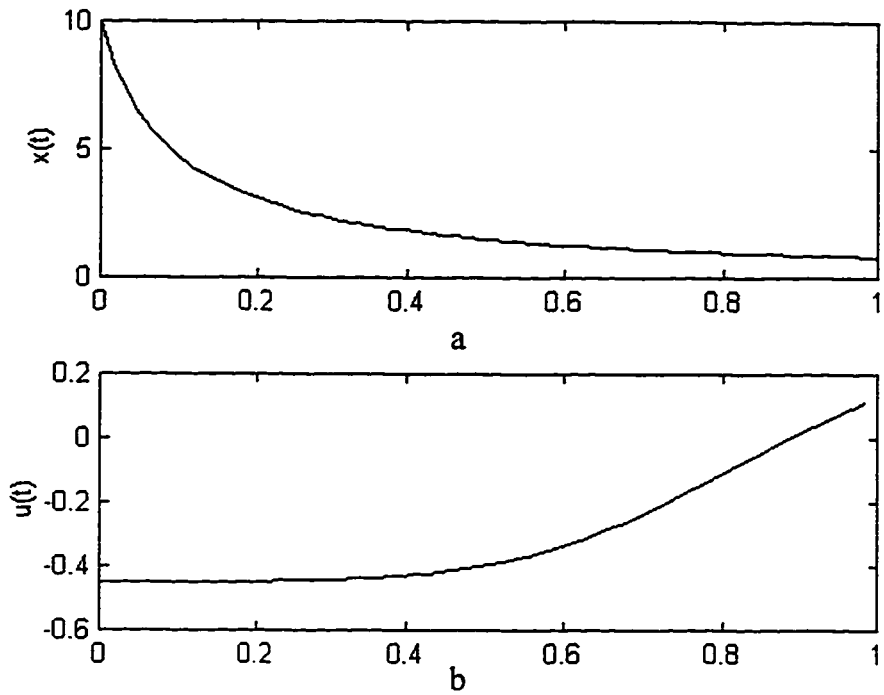


Figure 4.4 Optimal trajectory for Example 4.2. a. system state b. control effort.

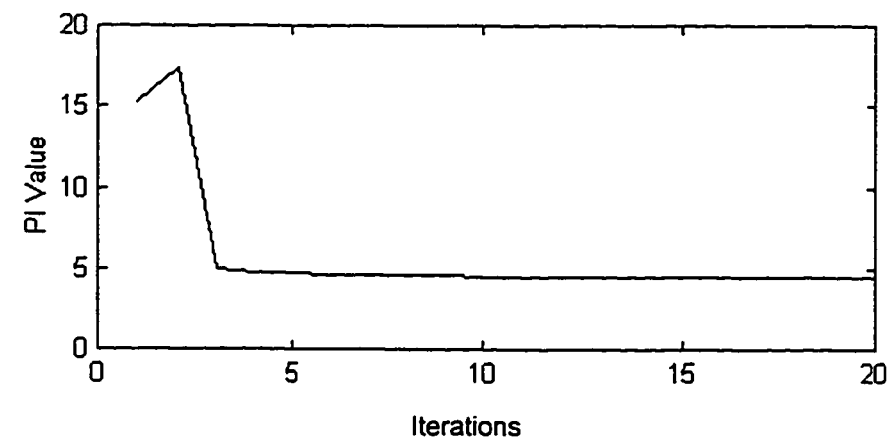


Figure 4.5 Decay of the PI vs. iterations for Example 4.2.

4.3 Shortest Distance between Two Points

In this example we consider the problem of finding the shortest length of the curve joining the two points $(a, x(a))$ and $(b, x(b))$. The length of the curve can be expressed by

$$\Im_L = \int_a^b \sqrt{1+(\dot{x}(t))^2}\ dt. \tag{4.7}$$

Replacing $\dot{x}(t)$ by $u(t)$ we have

$$\Im_L = \int_a^b \sqrt{1+u^2(t)}\ dt, \tag{4.8}$$

and

$$\dot{x}(t) = u(t). \tag{4.9}$$

The problem is discretized using the forward difference to have

$$\Im_L = T_s \sum_{k=0}^{N} \sqrt{1+u^2(k)} \tag{4.10}$$

and

$$x(k+1) = x(k) + T_S\, u(k) \tag{4.11}$$

where T_S is the step size and,

$N=(b-a)/T_S$.

To solve this problem using our approach, we add a penalty to the deviation of the

final state from a desired value x_b. The PI then becomes

$$\Im = T_s \sum_{k=0}^{N} \sqrt{1+u^2(k)} + \mathbf{H}\left(x(N) - x_b\right)^2. \qquad (4.12)$$

The gradient for this specific PI can be computed from (3.5) as

$$\frac{\partial^D \Im}{\partial W_i} = T_s \sum_{k=0}^{N} \left[\frac{u(k)}{\sqrt{1+u^2(k)}} \frac{\partial^D u(k)}{\partial W_i} \right] + 2\mathbf{H}\left(x(N) - x_b\right) \frac{\partial^D x(N)}{\partial W_i}. \qquad (4.13)$$

where the sensitivity derivatives $\frac{\partial^D x(k)}{\partial W_i}$ and $\frac{\partial^D u(k)}{\partial W_i}$ are computed from (3.11b) and (3.12) as before. The Jacobians of the plant are given by $A(k) = 1$ and $B(k) = T_S$.

The Matlab program is adapted to accommodate the new form of the PI and the following values of the parameters are used,

$T_S = 0.1$ $\quad$ $\mathbf{H} = 100$

$a = 1$ $\quad$ $x(a) \equiv x_a = 3$

$b = 5$ $\quad$ $x(b) \equiv x_b = 15.$

The MFNN used has two layers with 5 and 3 neurons respectively. When the training process started, the problem converged very quickly to the optimal trajectory which is the straight line connecting the initial and final points. The PI value obtained has been $\Im^* = 12.6494$ while the optimal theoretical value is $\Im_{opt} = 12.6491$. Figure 4.6 shows the state and control trajectories while Figure 4.7 shows the change of the PI versus iterations.

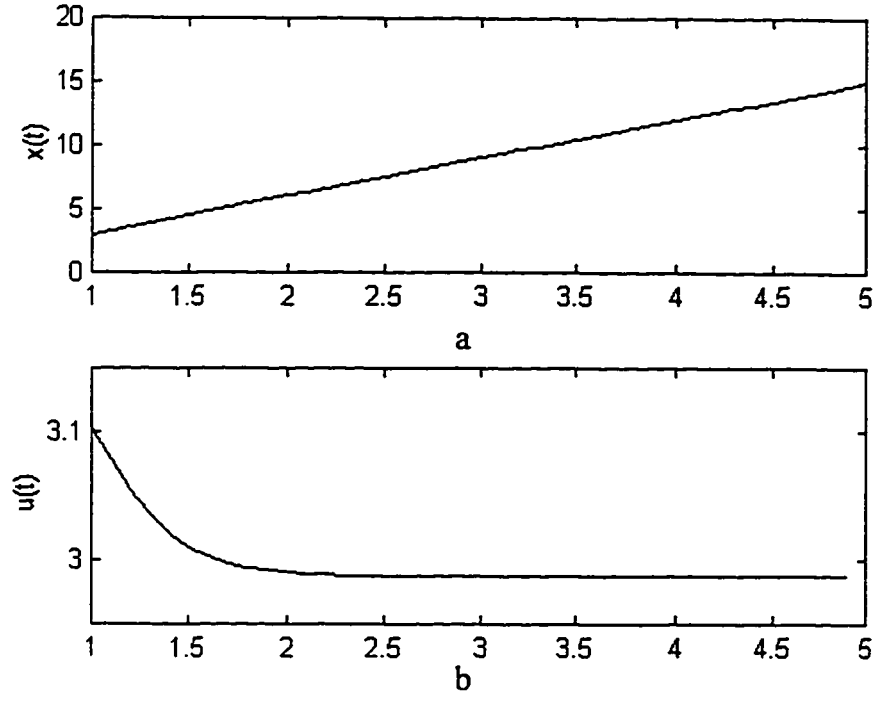


Figure 4.6 Optimal trajectory for Example 4.3.
a. system state b. control effort.

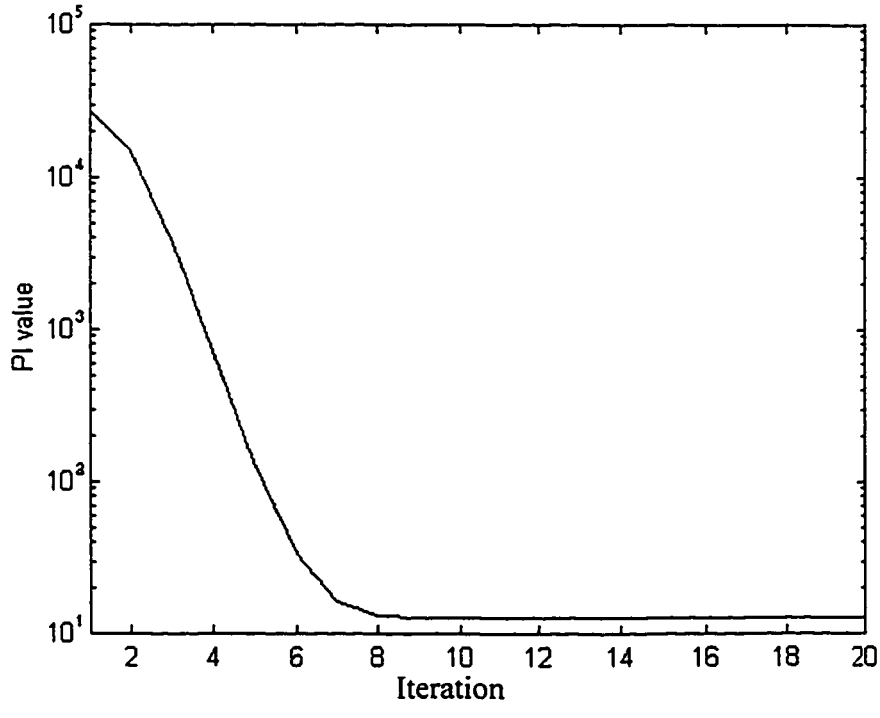


Figure 4.7 Decay of the PI vs. iterations for Example 4.3.

4.4 Stirred-Tank Chemical Reactor Problem

The stirred tank reactor is a vessel equipped with an agitator. Stirred tank reactors are classified into three types [14], batch reactors, continuous flow stirred tank reactors (CFSTR's) and semi-batch stirred tank reactors. The three types are illustrated in Figure 4.8. In batch reactors, the reactants are added to the empty vessel and the products are withdrawn after the completion of the reaction. In CFSTR's, the flow of the reactants and products into and out of the reactor is continuous. Finally in semibatch reactors, one of the reactants is charged initially to the reactor and the other reactants are fed continuously into the reactor. When the systems are well agitated , the properties of the reaction, such as temperature, pressure, and concentration, can be regarded to be uniform through out the reactor.

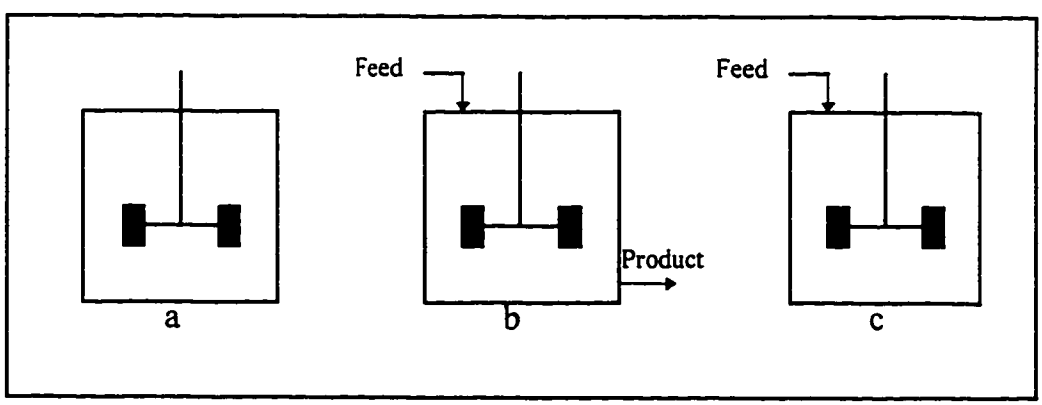


Figure 4.8 Three types of stirred tank reactors. a. batch b. continuous flow c. semi-batch.

Neural net-based optimal control is implemented to a continuous flow reaction process with a flow of a coolant through a coil in the reactor is controlling the reaction. The reaction is assumed to be first order, irreversible, and exothermic.

Two cases are considered for the type of control action applied, namely unconstrained and constrained control.

Case 1. Unconstrained Control

Problem Statement

The first order continuous flow reaction is described by the following two first-order deferential equations [14]:

$$\dot{x}_1(t) = -2[x_1(t) + 0.25] + [x_2(t) + 0.5]\exp\left[\frac{25x_1(t)}{x_1(t)+2}\right] - [x_1(t) + 0.25]u(t) \quad (4.14)$$

$$\dot{x}_2(t) = 0.5 - x_2(t) - [x_2(t) + 0.5]\exp\left[\frac{25x_1(t)}{x_1(t)+2}\right] \quad (4.15)$$

The system state $x_1(t)$ represents the deviation from the steady-state temperature $T(t)$ and $x_2(t)$ is the deviation from the steady-state concentration $C(t)$. $u(t)$ is a normalized control signal which represents the effect of the coolant flow on the reaction.

The initial states of the system are x(0)=[0.05 0]^T. The objective of this problem is to maintain the temperature and the concentration of the reaction close to their steady-state values and at the same time minimize the control effort for $t \in [0, 0.78]$. Thus, the performance index can be written as,

$$\mathfrak{I} = \int_0^{0.78} [x_1^2(t) + x_2^2(t) + ru^2(t)]dt \quad (4.16)$$

with the weighting factor r is selected arbitrarily as 0.1.

Problem Solution

To solve this problem using our approach, the problem is discretized using the forward difference to have,

$$x_1(k+1) = x_1(k) + T_s\left\{-2[x_1(k)+0.25] + [x_2(k)+0.5]\exp\left[\frac{25x_1(k)}{x_1(k)+2}\right] - [x_1(k)+0.25]u(k)\right\} \tag{4.17}$$

$$x_2(k+1) = x_2(k) + T_s\left\{0.5 - x_2(k) - [x_2(k)+0.5]\exp\left[\frac{25x_1(k)}{x_1(k)+2}\right]\right\} \tag{4.18}$$

and

$$\Im_k = \sum_{k=0}^{N} T_s[x_1^2(k) + x_2^2(k) + ru^2(k)] \tag{4.19}$$

where , as before, N is the number of time steps and T_S is the step size

The Jacobian matrices defined by equations 3.8 and 3.9 are,

$$A(k) = \begin{bmatrix} a_1(k) & a_2(k) \\ a_3(k) & a_4(k) \end{bmatrix}$$

with

$$a_1(k) = 1 + T_s\left\{-2 + [x_2(k)+0.5]\frac{50}{(x_1(k)+2)^2}\exp\left[\frac{25x_1(k)}{x_1(k)+2}\right] - u(k)\right\} \tag{4.20}$$

$$a_2(k) = T_s\exp\left[\frac{25x_1(k)}{x_1(k)+2}\right] \tag{4.21}$$

$$a_3(k) = -T_s[x_2(k)+0.5]\frac{50}{(x_1(k)+2)^2}\exp\left[\frac{25x_1(k)}{x_1(k)+2}\right] \tag{4.22}$$

$$a_1(k) = 1 - T_s\left\{1 + \exp\left[\frac{25x_1(k)}{x_1(k)+2}\right]\right\} \qquad (4.23)$$

and

$$B(k) = \begin{bmatrix} b_1(k) \\ b_2(k) \end{bmatrix}$$

with

$b_1(k) = -T_S[x_1(k) + 0.25]$; $b_2(k) = 0.$

From equation 4.19, the weighting matrices $Q(k)$ and $R(k)$ are,

$Q(k) = T_S$; $R(k) = T_S\, r.$

The problem is solved using the developed Matlab program. The training process converged in 27 iterations. The number of sampling steps N has a significant effect on the number of iterations needed for convergence. As the value of N is made larger, the problem converges in less number of iterations. During training, the number of steps N is set to 99 for the first 26 iterations. When N is changed to 430, the problem converged to the optimal solution immediately in the 27th iteration. For comparison, the value of the performance index obtained is shown in Table 4.3 together with those obtained via other numerical open loop techniques [19]. The results obtained by the MFNN are similar to those obtained by other numerical techniques with the added advantage of providing the control signal in a state feedback form. The trajectories of the state and control signals obtained are shown in Figures 4.9 and 4.10.

Technique		Performance Index
MFNN	State-Feedback Control	0.02680
open loop control	Steepest Descent	0.02680
	Variation of Extremals	0.02660
	Quasilinearization	0.02660
	Gradient Projection	0.02725

Table 4.3 Solution of the unconstrained stirred-tank control problem.

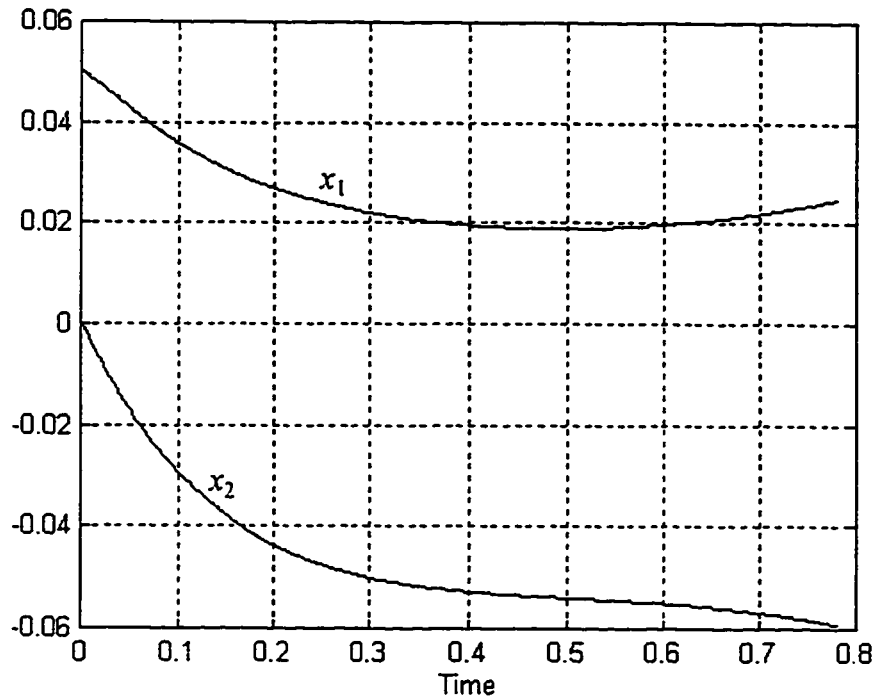


Figure 4.9 States of the unconstrained stirred-tank reactor problem using the MFNN state-feedback control.

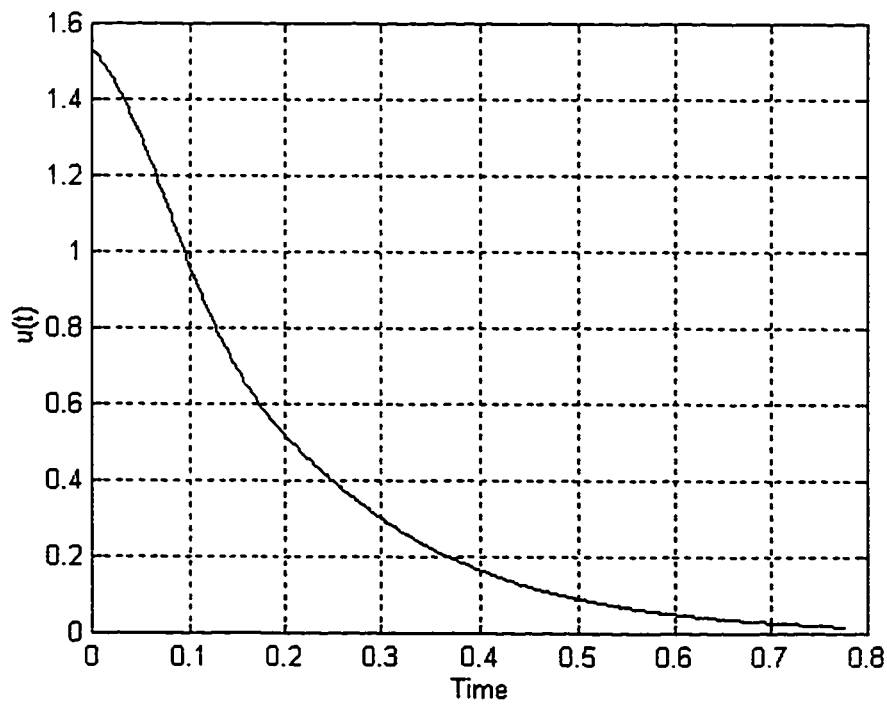


Figure 4.10 The control signal for the unconstrained stirred-tank reactor problem using the MFNN state-feedback control.

Case 2. Constrained Control

Problem Statement

In this case, the performance index is

$$J = \int_0^{0.78} [x_1^2(t) + x_2^2(t)]dt \qquad (4.24)$$

and the control signal is required to satisfy the constraint,

$$-1.0 \le u(t) \le 1.0, \qquad t \in [0,0.78]. \qquad (4.25)$$

Furthermore, the final values of the system states are required to be at the origin. In other words,

$$X(0.78) = X(N) = 0$$

Problem Solution

Slight modification is made to the computer program in order to accommodate the change in the problem formulation. A saturation statement is added to the program to limit the control signal between -1 and +1. When these limits are exceeded, the control signal is truncated and the partial derivatives of the control signal with respect to other variables are set to zero. To force the final values of the states to be at the origin, the final value of the weighting matrix $Q(N)$ is set to comparably high value. In this case, $Q(N)$ is set to $10I_2$, while $Q(k)$, k=0, ..., N-1 is set to I_2.

To obtain good initial weights, the training is first conducted without limit on the control signal. Then, the limits are imposed on the control signal and the training is resumed. This causes the solution to converge to a minima similar to that obtained using the Gradient Projection algorithm [19]. The values of the performance index are shown in Table 4.4 for comparison. In addition, the final trajectories of the state and control are shown in figures 4.11 and 4.12.

Technique	**Performance Index**
MFNN State-Feedback Control	0.0023
Gradient Projection (open loop)	0.0022

Table 4.4 Solution of the constrained stirred-tank control problem.

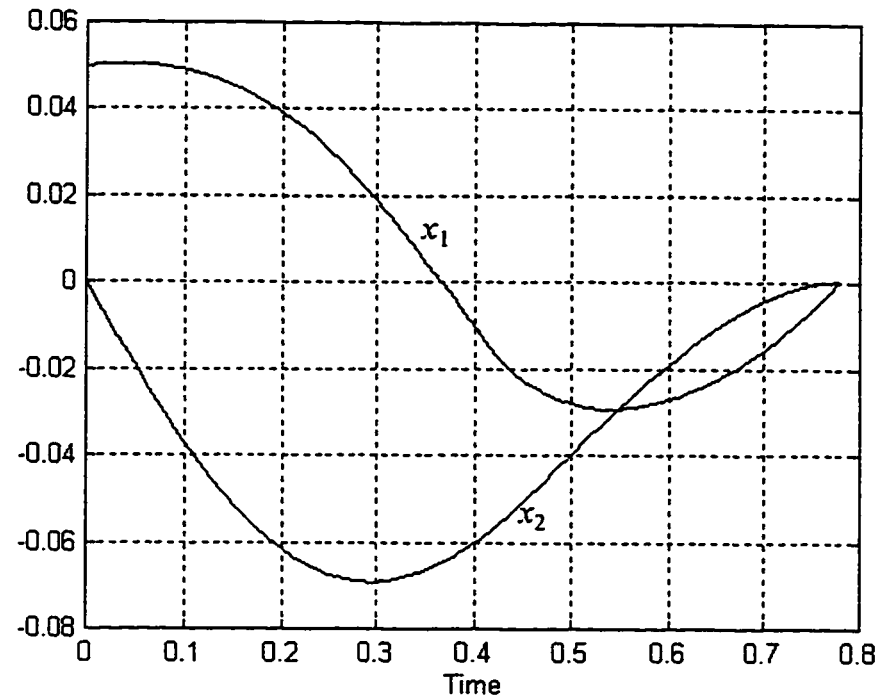


Figure 4.11 States of the constrained stirred-tank reactor problem using the MFNN state-feedback control.

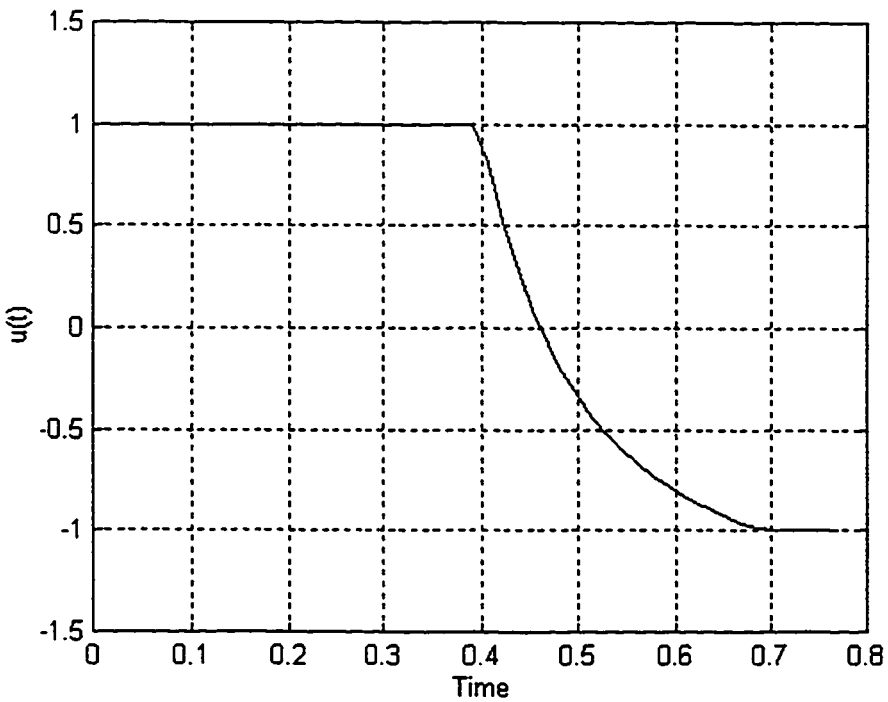


Figure 4.12 The control signal for the constrained stirred-tank reactor problem using the MFNN state-feedback control.

4.5 A Second Order Nonlinear Plant

Consider the following second-order nonlinear continuous plant,

$$\dot{x}_1(t) = x_2(t) \tag{4.26}$$

$$\dot{x}_2(t) = a(b - x_1^2(t))x_2(t) + x_1(t) + cu(t) \tag{4.27}$$

where a, b, and c are the plant parameters. The discrete form of this system is obtained using the backward difference as,

$$x_1(k) = x_1(k-1) + T_s x_2(k-1) \tag{4.28}$$

$$x_2(k) = x_2(k-1) + T_s\left[a\left(b - x_1^2(k-1)\right)x_2(k-1) + x_1(k-1) + cu(k-1)\right], \tag{4.29}$$

where, again, T_s is the step size. The system parameters are chosen as a=1, b=1, and c=1. The initial values of the system states are chosen as,

$$x(0) \equiv x_0 = \begin{bmatrix} 1 & 0 \end{bmatrix}^T.$$

This plant is used to demonstrate how the developed MFNN control scheme can be used to solve optimal tracking, regulation, terminal control, minimum control effort, minimum time, and output tracking problems. In some of the problem considered, the variable adaptation algorithm described in section 3.2.2 is used. In other problems, constant or piecewise-constant adaptation rates are used. The neural network used has two hidden layers with ten and three neurons respectively.

4.5.1 *Optimal Tracking*

The objective here is to find the control signal $u(k)$ that minimizes the quadratic PI given by equation 3.13 with,

$$x_d(k) \equiv \begin{bmatrix} x_{1d}(k) \\ x_{2d}(k) \end{bmatrix} = \begin{bmatrix} \sin 0.2k \\ 0 \end{bmatrix},$$

$$Q(k) = \begin{bmatrix} 1.5 & 0 \\ 0 & 0 \end{bmatrix} \quad \forall k,$$

$$R(k) = 0 \quad \forall k.$$

It can be observed from the choice of $Q(k)$ that the objective was only to force state $x_1(k)$ to follow a sinusoidal trajectory. The values of N and T_s have been chosen as 49 and 0.1 respectively. From the plant equations, the Jacobian matrices $A(k)$ and $B(k)$ are derived as,

$$A(k) = \begin{bmatrix} 1 & T_s \\ -2aT_s x_1(k-1)x_2(k-1) + T_s & 1 + aT_s\left(b - x_1^2(k-1)\right) \end{bmatrix}$$

$$B(k) = \begin{bmatrix} 0 \\ T_s\, c \end{bmatrix}.$$

The normalized time τ was taken as the actual time between 0 and 4.9 seconds. The adaptation scheme described in section 3.2.2 was used with,

$1.95 < \mu_i < 5$

$\mu_0 = 0.5$

$\mu_f = 2$

$\mu_r = 0.95.$

Figure 4.13 shows the states and desired states of the system after 640 iterations. The figure shows clearly that the MFNN has been successfully trained to force one of the system states to follow "optimally" a desired response. The trend of the PI value over iterations is shown in Figure 4.14.

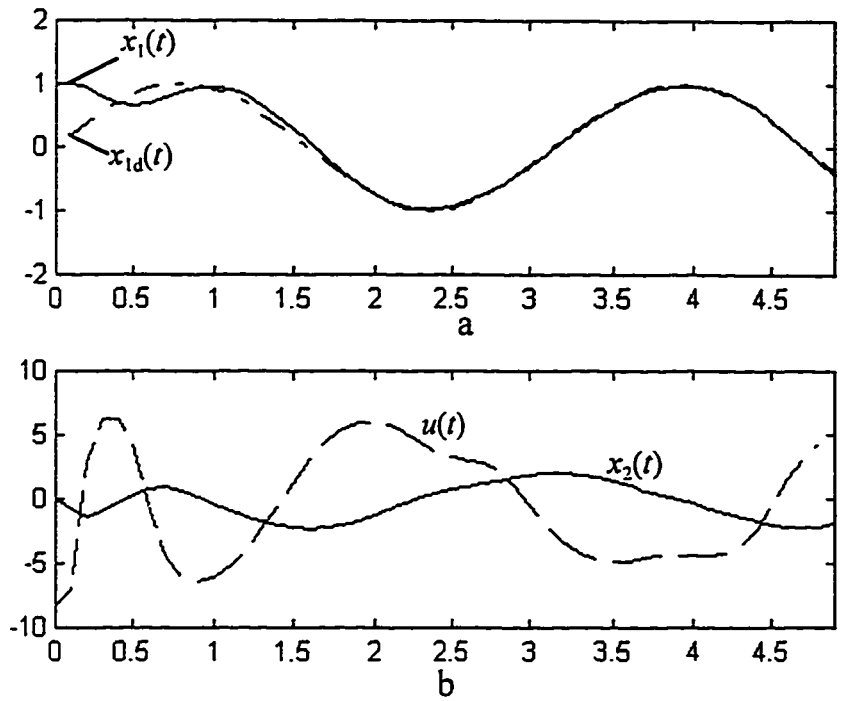


Figure 4.13 Optimal trajectories of the tracking problem after 640 iterations. a. $x_1(t)$ tracking a desired sine wave. b. $x_2(t)$ and $u(t)$

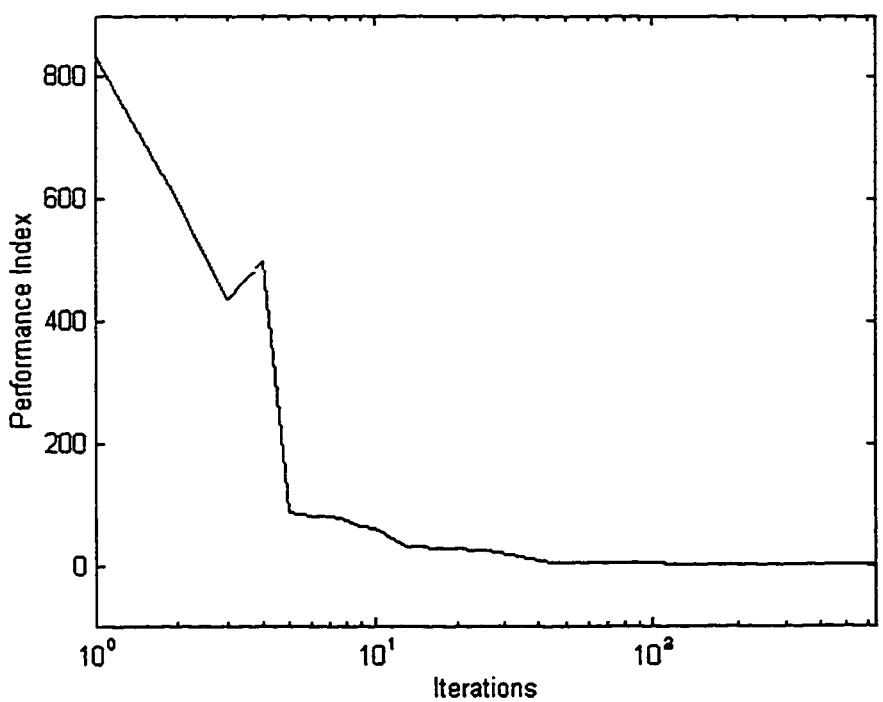


Figure 4.14 Decay of the performance index over iterations for the tracking problem.

4.5.2 Regulation Problem

In this case, the values of $Q(k)$ and $R(k)$ are non-zeros for all k and the objective now is to regulate the system states near the zero value with moderate control effort as specified by a non-zero weight $R(k)$. The desired states are set to zero for all k. $Q(k)$ and $R(k)$ are chosen as,

$$Q(k) = 10\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \forall k;$$

$$R(k) = 0.1 \quad \forall k.$$

The values of N and T_S have been chosen as 30 and 0.1 respectively. Same adaptation parameters as in the tracking problem are used here. The computed optimal states and control effort are shown in Figure 4.15. The weight imposed on the input limits its freedom to regulate the states of the system and the result is a trade-off between the regulation of the states and the amount of control effort applied. The PI trend is shown in Figure 4.16.

For comparison, the training is restarted with the value of $R(k)$ increased to 10 $\forall k$. This imposes more penalty on the control effort applied. The states and control effort after training are shown in Figure 4.17. As expected, the amplitude of the control signal is smaller while the states are less regulated than the case when $R(k)$=0.1.

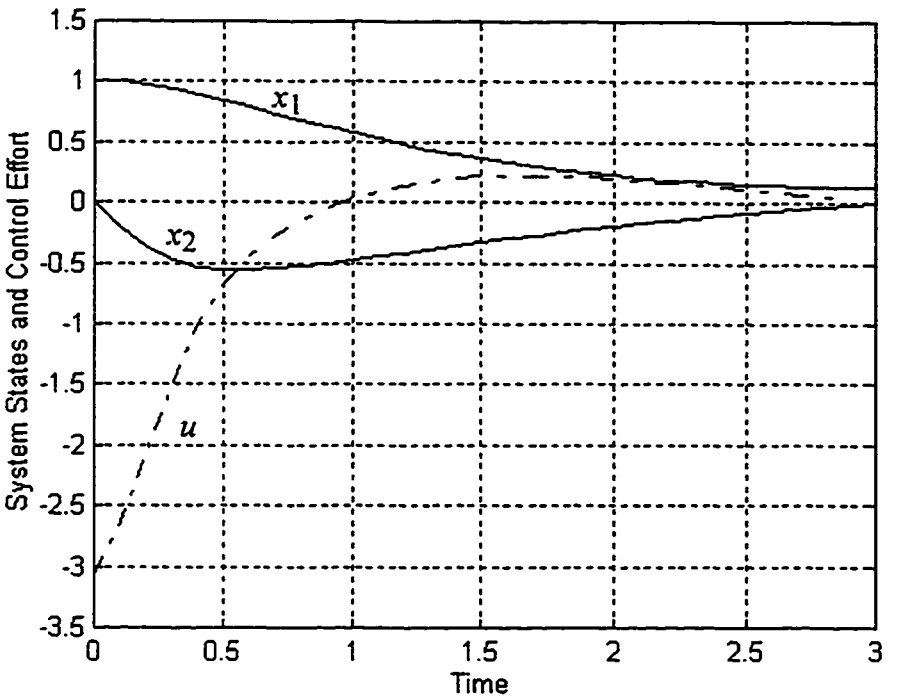


Figure 4.15 The states of the system and the control effort for the regulation problem after 100 iterations ($R(k)$=0.1).

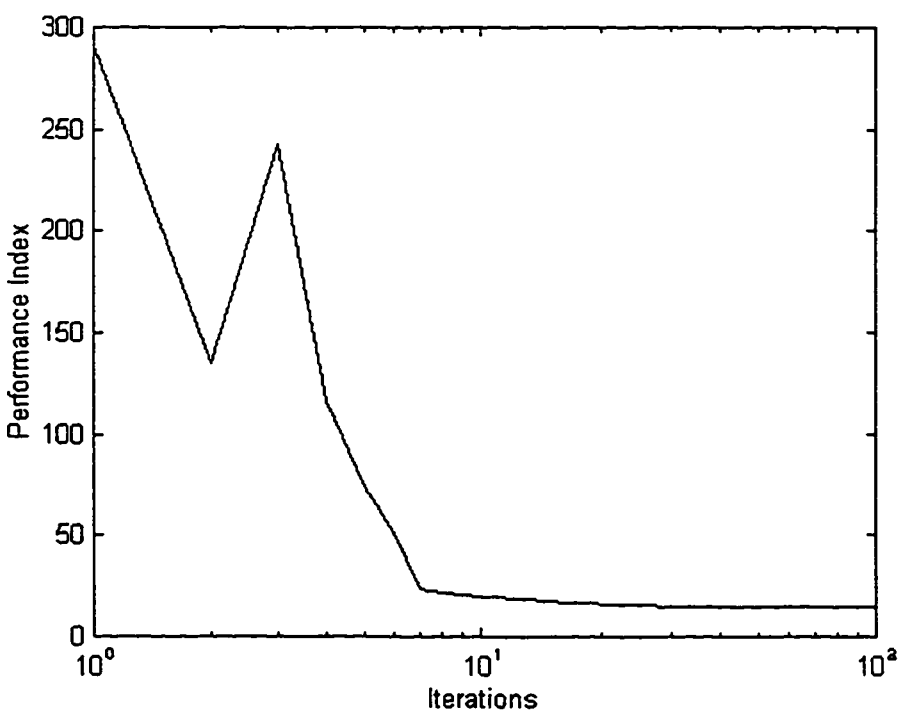


Figure 4.16 Decay of the performance index over iterations for the regulation problem.

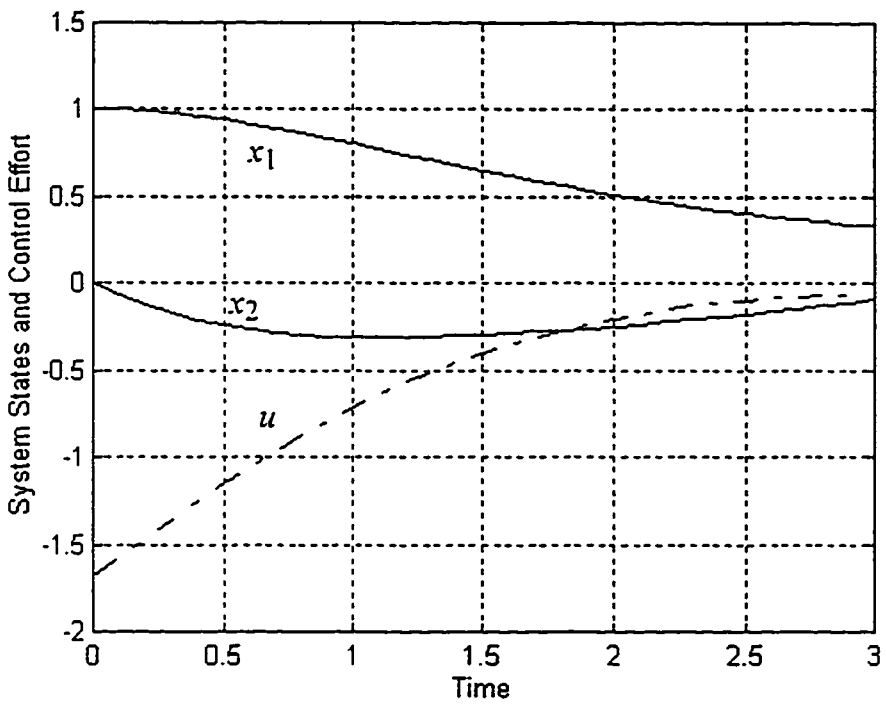


Figure 4.17 The states of the system and the control effort for the regulation problem with $R(k)$=10.

4.5.3 *Terminal Control*

The following values are chosen for the desired terminal states and the weighting matrices,

$$x_d(k) = \begin{cases} \mathbf{0} & \text{for } k < N \\ \begin{bmatrix} -0.5 \\ 0.5 \end{bmatrix} & \text{for } k = N \end{cases}$$

$$Q(k) = \begin{cases} \mathbf{0} & \text{for } k < N \\ 1000 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \text{for } k = N \end{cases}$$

$$R(k) = 0 \quad \forall k .$$

where a bold zero indicates a zero vector. The adaptation rate is chosen to be constant value, equals to 0.001. T_s and N are set to 0.05 and 60 respectively. Again, τ is chosen to be between 0 and 3 representing the true time. The final states converged exactly to the desired values in relatively few number of iterations. The states and control effort for this case are shown in Figure 4.18 and the trend of the performance index is shown in Figure 4.19.

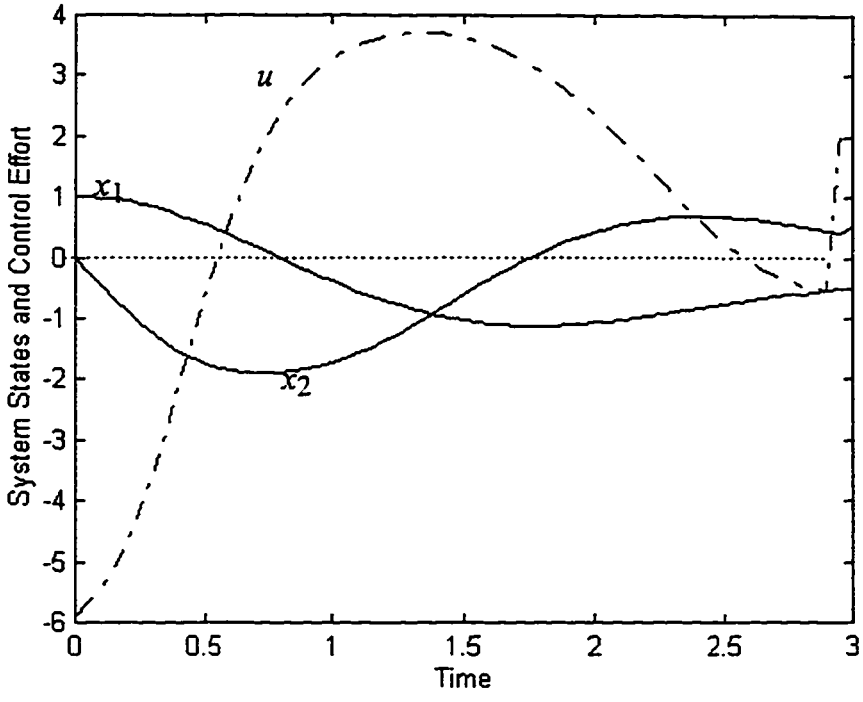


Figure 4.18 The states of the system and the control effort for the terminal control problem after 40 iterations.

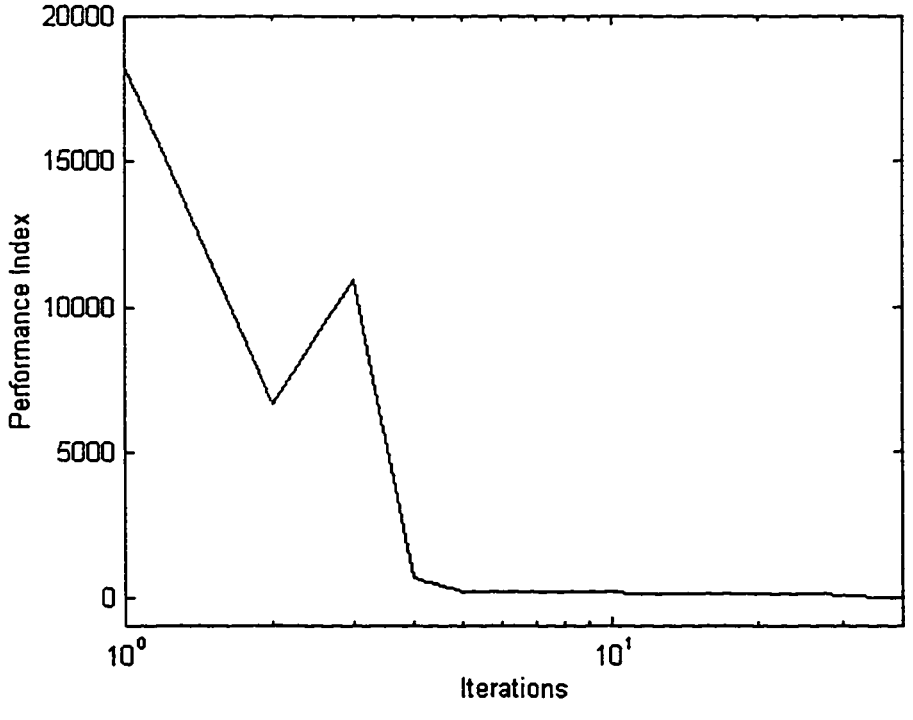


Figure 4.19 Performance index over iterations for the terminal control problem.

4.5.4 Minimum Control Effort

The system states for this case are required to reach the origin at the final fixed time with minimum control effort. Both minimum fuel and minimum energy problems have been considered in the simulation.

Minimum Fuel Problem: Due to the existence of the absolute value term in the PI,

$$\Im = \sum_{k=1}^{N} |u(k)|_{R(k)}$$

the minima is expected to be relatively sharp. This makes the oscillation of the PI value around its minima during training larger than that in problems involving quadratic PI's. The amount of oscillation can be reduced by decreasing the adaptation rate. For the problem under study, the training is started with the adaptation rate equal to 0.01 to speed up the convergence. When the PI value becomes closer to the minima, the adaptation rate is decreased to 0.001 to suppress the oscillation and to fine-tune the final values of the states close to the origin. T_S and N remain as before (T_S=.1 and N=29). The weighting matrices are set to,

$$Q(k) = \begin{cases} \mathbf{0} & \text{for } k < N \\ 10\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \text{for } k = N \end{cases}$$

$R(k) = 0.1 \quad \forall k$.

The final states and control effort are shown in Figure 4.20. The value of the PI is shown in Figure 4.21 where it is clear that when the adaptation rate is decreased at iteration 68, the amount of oscillation is reduced.

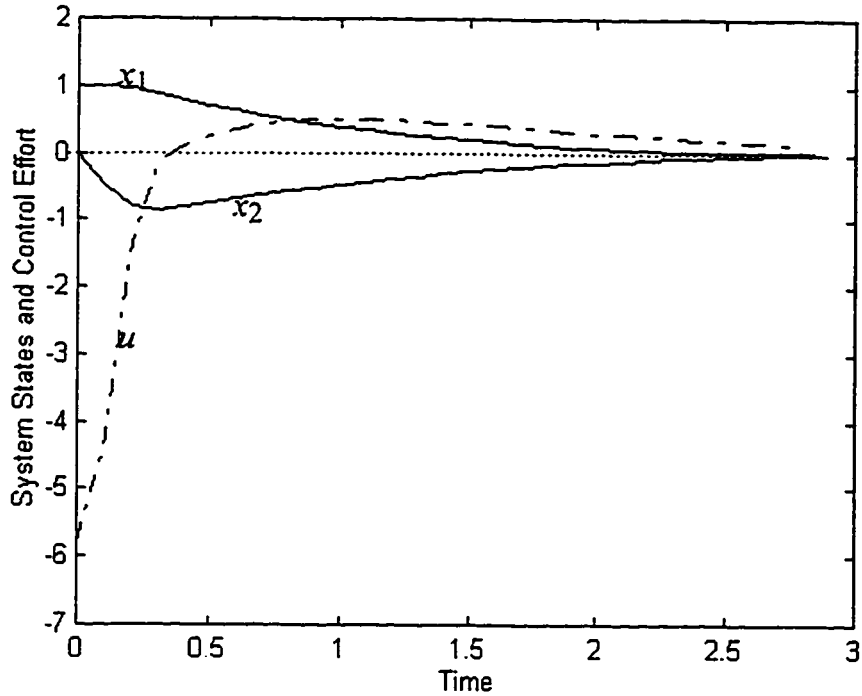


Figure 4.20 The states of the system and the control effort for the minimum fuel problem after 180 iterations.

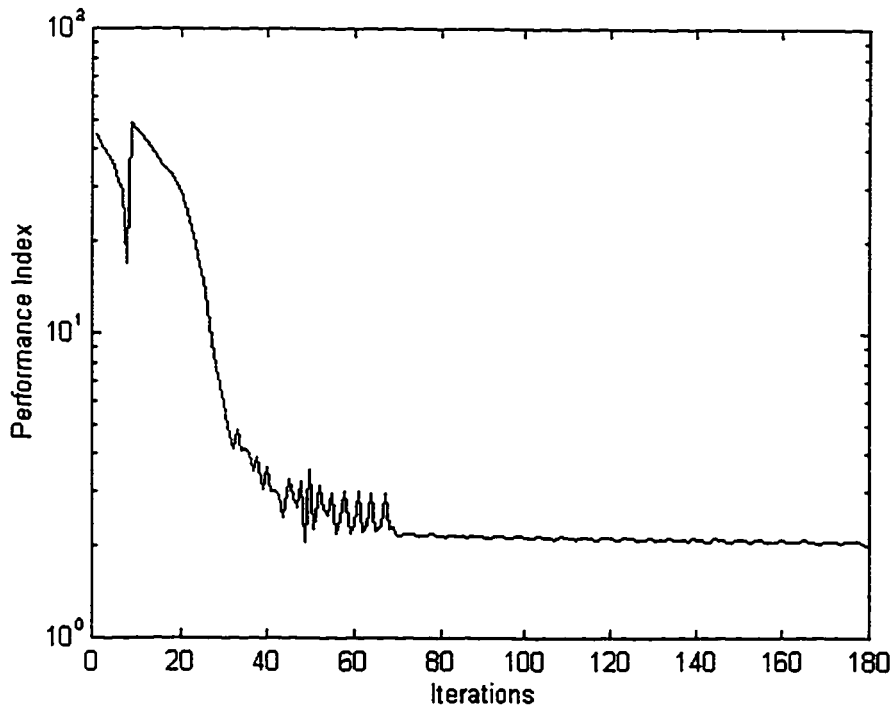


Figure 4.21 Decay of the performance index over iterations for the minimum fuel problem.

Minimum Energy Problem: The neural network is trained to force the final states of the system to the origin with minimum energy dissipation. The values of N and T_S used have been 29 and 0.1 respectively. The adaptation rate is fixed to 0.001 and the weighting matrices are chosen as,

$$Q(k) = \begin{cases} \mathbf{0} & \text{for } k < N \\ 500\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \text{for } k = N \end{cases}$$

$$R(k) = 0.1 \quad \forall k .$$

The problem converged and the states and control effort after 1000 iterations are shown in Figure 4.22. The decay of the PI value over iterations is shown in Figure 4.23. The By comparing Figures 4.20 and 4.22 we find that the final states for the minimum fuel problem are closer to the origin and the maximum amplitude of the control effort for the minimum energy problem is less. This is expected since the quadratic PI imposes less penalty on the states and control effort when their values are less than one but imposes larger penalty when their values are grater than one.

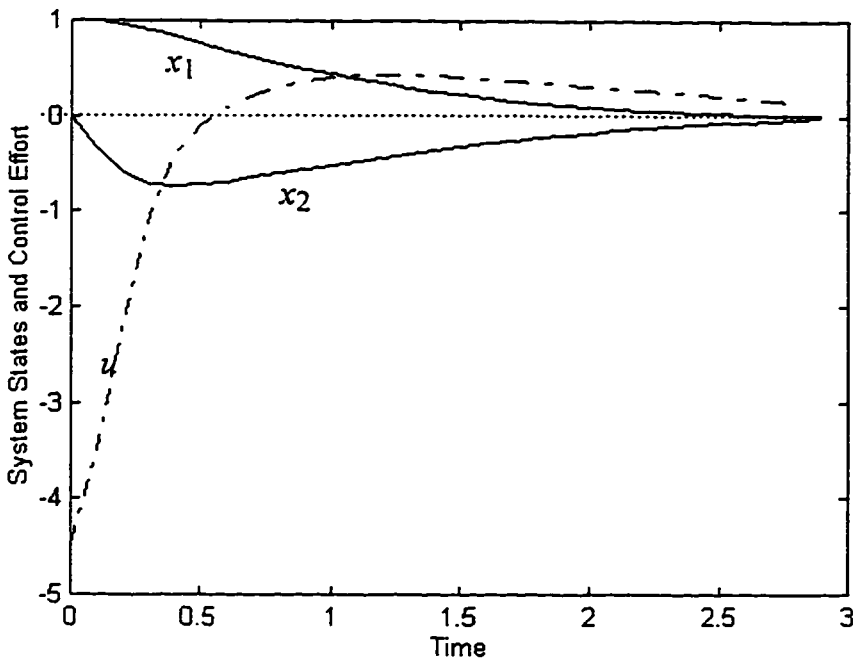


Figure 4.22 The states of the system and the control effort for the minimum energy problem after 1000 iterations.

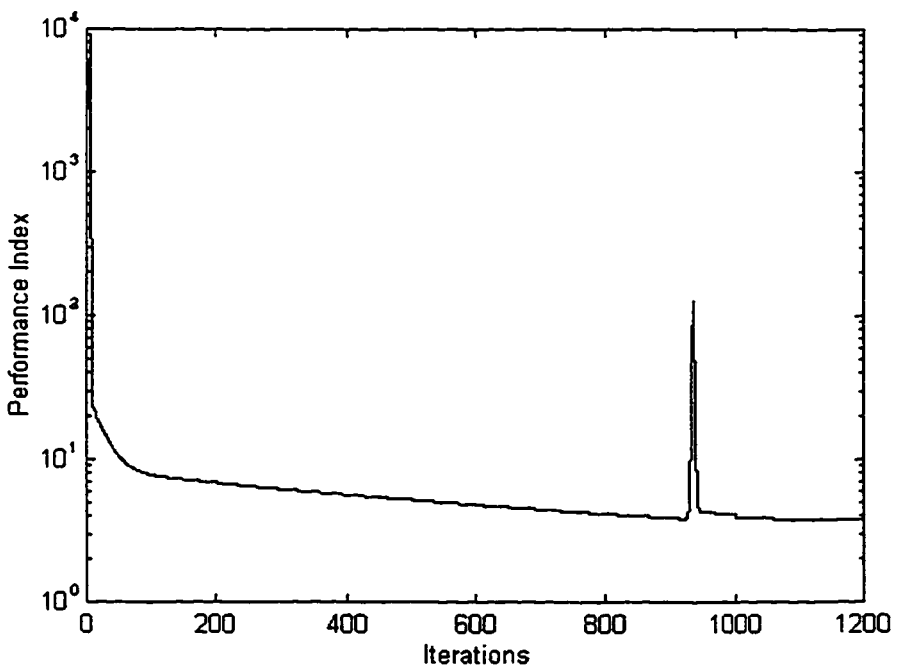


Figure 4.23 The performance index over iterations for the minimum energy problem.

4.5.5 Minimum Time Problem

The objective in this case is to find out the control effort that will drive the system to the origin in minimum time. The weighting matrices are chosen as,

$$Q(k) = \begin{cases} \mathbf{0} & \text{for } k < N \\ 200 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \text{for } k = N \end{cases}$$

$R(k) = 0 \quad$ for all k.

The control signal is bounded by,

$-5.0 < u(k) < 5.0.$

The approach proposed in section 3.3.4 is used here with the values of γ and β set initially to 0.1 and 0.0 respectively. The other parameters are chosen as,

N=30 $\qquad$ δ = 10E-3

ρ = 10E-6 $\qquad$ i_{max}=100

The sampling time converged finally to a value of 0.0344 as shown in Figure 4.24. Thus, the optimal time achieved by the network is $t_f^* = NT_S^* = 1.032$ seconds. The final trajectory of the system states together with the control effort are shown in Figure 4.25.

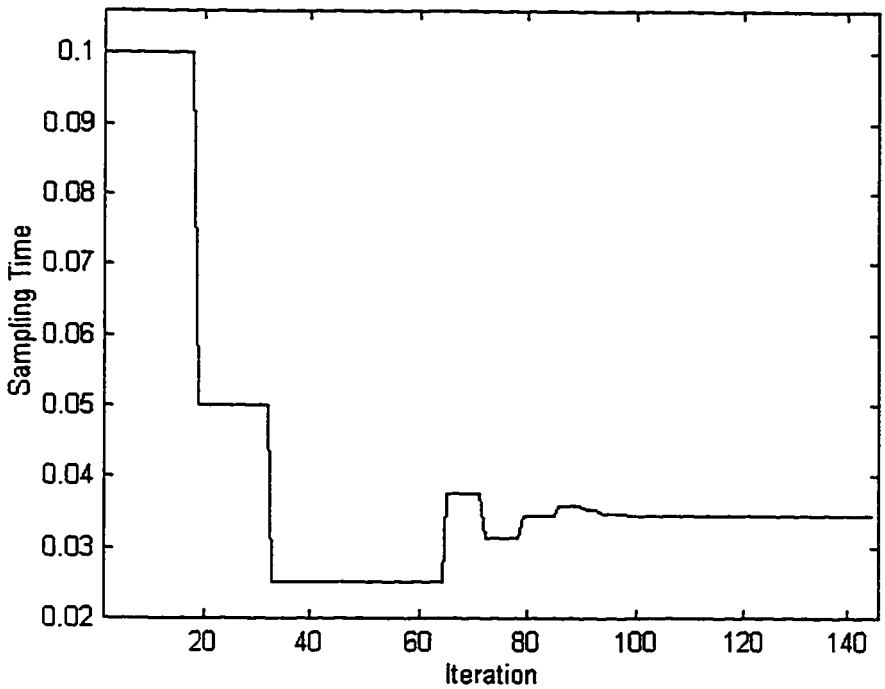


Figure 4.24 The sampling time during training.

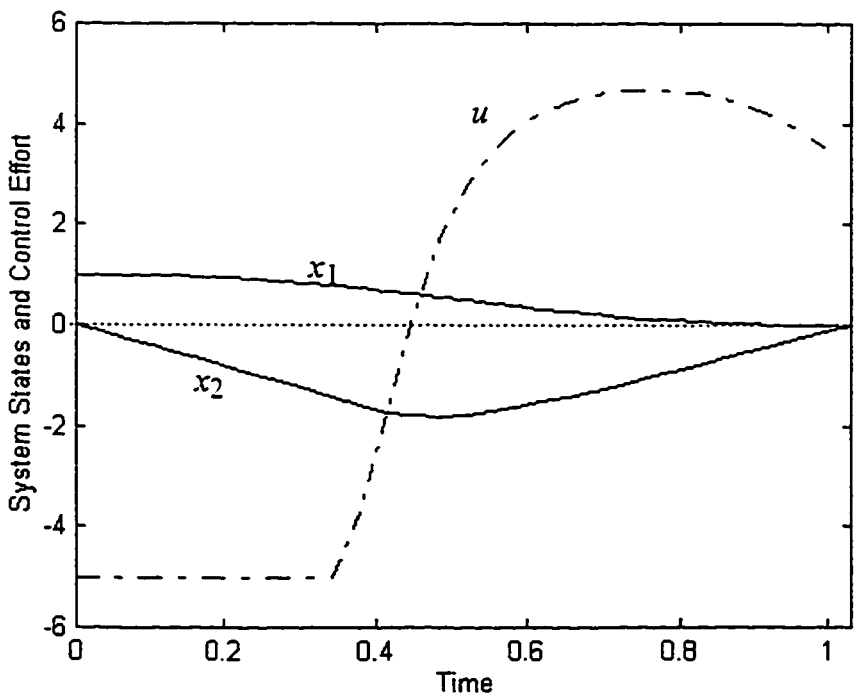


Figure 4.25 Final trajectories of the system states and control effort of the time optimal control problem.

4.5.6 Output tracking problem (with integrator):

The nonlinear plant described by equations (4.28) and (4.29) also has been used for output tracking. The output equation has been taken as

$$y(k) = \begin{bmatrix} 1 & 0 \end{bmatrix} x(k)$$

It is assumed that the plant will experience a step-like setpoint in the range of ±3. Therefore, the integrator order l and $\alpha(q)$ have been taken as 1 and q^{-1} respectively. Further the training setpoint has been generated as step changes in 20 different uniformly distributed random levels within the range ±3 as shown in Figure 4.26**a**.

However, we failed to train the neural controller to follow this setpoint in one shot. In order to achieve satisfactory training, we conducted the training in four stages. In the first stage, the controller has been trained to bring about the plant output follow one cycle of a square wave of amplitude one. The neural weights obtained were used as the initial weight in the second stage where the controller has been trained for two cycles of a square wave having the same amplitude as earlier. In the third stage the amplitude of the square wave has been raised to three. In the final stage, the setpoint has been the random step changes as shown in Figure 4.26**a**, which also shows the plant output at the end of training. Figure 4.26**b** shows the control implementation for a typical setpoint that was not part of the training setpoint. Both Figures 4.26**a** and 4.26**b** correspond to the minimization of the PI given by equation (3.28) with

$$N = 2000: \quad M(k) = 10, \underline{Q}(k) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, R(k-1) = 0.01, \forall k;$$

$\beta(q)$ has been chosen as $1-q^{-1}$, which penalizes the plant input variation instead of the input amplitude.

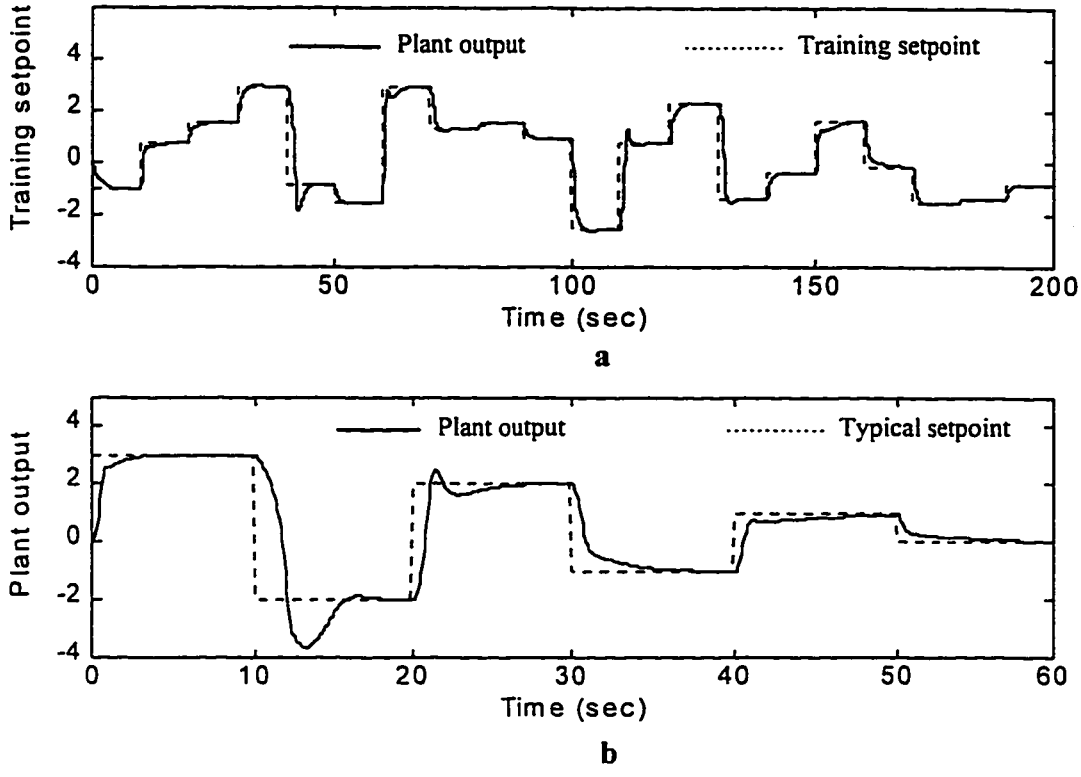


Fig. 4.26 Output tracking with integrator a. Training setpoint b. Typical setpoint

4.6 Robustness Issues

Since the optimal control signal provided by the MFNN is in a state feedback form, the sensitivity of the control system to initial states and parameter variation is expected to be less than that for the open loop optimal control [38]. The robustness of the neural optimal control can be further increased utilizing the

memorization and interpolation features of the neural networks. The neural network can be trained on a domain of initial states and system parameters. It can then produce acceptable control signals for initial states and system parameters that lie within this domain.

In this section, the robustness of the MFNN state feedback optimal control to change of initial states and system parameter is investigated. The second order nonlinear system described in the previous section will be used for the illustration.

4.6.1 Changes in Initial States

Consider the system described in the previous section. A set of initial values of the state x_1 are picked from a normal distribution with a mean of 1.0 and a standard deviation of 0.5. An MFNN is trained to minimize an aggregated PI including all the initial states in the set. Another network is trained only for $x_1(0)=1$. Both networks are then used to control the system with $x_1(0)=0.2$. Table 4.5 shows the performance of each network in terms of their PI value. The optimal value obtained by training the network with $x_1(0)=0.2$ is also included for comparison. As we expected, training the neural networks with random set of initial states results in a closer behavior to the optimal one.

Initial State $x_1(0)$ Used in Training	**PI value while control with $x_1(0)=0.2$**
1.0	0.1665
normal distribution	0.0505
0.2	0.0446 (reference)

Table 4.5 PI values for training with three different sets of initial states.

4.6.2 Changes in System Parameters

A similar experiment is conducted on the uncertainty of system parameters. The neuro-controller is trained on a set of system parameters. When trained, the neural network should produce acceptable response for any value inside this set. Our nonlinear system has three parameters appearing in its dynamic equations, namely a, b, and c. The parameter a is chosen to be varied while the rest are kept constant. In each training iteration, the value of a is varied according to a continuous uniform distribution with a range between 0.0 and 6.0. After 250 iterations, the training is terminated and the weights of the MFNN are stored. For comparison, another network is trained on a fixed value of 1.0 for the parameter a and the resulting weights are stored. The two networks are then used to control the system with the value of a is set to 5.5. As expected, the network trained on a range of the parameter a result in a much closer PI value to the optimal one than that obtained by the other network. Table 4.6 show a numerical comparison between the PI values obtained by the two networks.

Value of the parameter a Used in Training	**PI value while control with a=5.5**
1.0	289.849
uniform distribution	1.4004
5.5	1.1062 (reference)

Table 4.6 PI values for training with three different values of the parameter a.

4.7 Computational Experience

From the examples attempted in this chapter, we have noticed the importance of choosing some of the training parameters on the MFNN Training. Poor choice of these parameters may lead to undesirable results such as divergence of the training process, slow convergence, convergence to local minimum, or weight oscillation. In this section, the effect of these parameters on the training process is elaborated.

4.7.1 Effect of Number of Time Steps

As the number of time steps N increases, the computation time per iteration obviously will increase. For example, when N is set to 61 the computation time per iteration for the problem considered in section 4.2 has been 9.44 seconds on a 486/66 PC. When N was increased to 100, the computation time increased to 15.0 seconds. On the other hand, it was found that increasing N usually decreases the number of iterations required by the algorithm to converge. We have seen an example of that in the unconstrained control of the stirred tank reactor problem. When N was increased from 99 to 430 at the 26^{th} iteration, the algorithm converged in only one more iteration. The value of N also affects the stability of the training process. From some of the cases faced, the weights might diverge because the value of N has been too large which results in a large value of the gradient $\partial^{(p)}\mathfrak{I}/\partial W_i$. This problem was resolved by starting with small N and then gradually increasing it until the desired N is reached.

4.7.2 Effect of number of neurons of the MFNN

The number of neurons of the MFNN affects the training process in two wayes. First, as the number of neurons increases the computation time per iteration duration will increase. Second, changing the number of neurons will affect the shape of the PI over the weight space and thus may affect the speed and accuracy of convergence. In our study, the number of neurons is chosen arbitrarily large enough to give the MFNN the freedom in locating the optimal solution of the problem.

4.7.3 Effect of Adaptation Rate

As pointed out in section 3.2.2, adaptation rate η is found to be a critical factor affecting the stability, weight oscillation, and speed of convergence of the training process. In almost all of the problems considered in this chapter, the training process diverges when a large value of η is used. This was not a big obstacle for us because of two reasons, first training is conducted off line and secondly the number of iterations for convergence, in most of the problems, is comparably low allowing for the use of different values of η. One possible way to deal with this situation is to use an adapted η. The value of η should be decreased when the weight oscillation is large and increased when the convergence is slow. This scheme has been used in the *minimum fuel problem*. The reader may refer to section 4.5.4.

The selection of the adaptation rate η is also affected by the weighting matrices of the PI. Very large values of the weighting matrices may cause the training to

diverge. In this case, the adaptation rate could be decreased as a compromise. This scheme has been used in the *shortest distance problem* described in section 4.3. When we started with **H**=1 and η=0.1, the desired final state could not be reached. Then, we increased **H** to 100 which caused the problem to diverge. To overcome this problem, η was decreased to 0.0001 and the problem converged to the expected solution.

4.7.4 Effect of Initial Weights of the MFNN

In our algorithms, the choice of the initial weights of training the MFNN plays an important role on the stability of training, number of iterations for convergence and the quality of the minimum obtained. The choice of the initial weights has been made randomly. In some of the problems attempted, large initial weights resulted in a large input signal causing the training process to diverge. This problem was faced in example 4.2 and was resolved by decreasing the amplitude of the initial weights. From another view point, there could be many minima in the weight space and thus the choice of the initial weights dictates the convergence to either of these minima. One way followed to overcome this problem is to try different initial weights and then choose the ones resulting in the smallest value of the PI.

CHAPTER V

CONCLUSIONS AND RECOMMENDATION FOR FUTURE WORK

5.1 General Observations and Conclusions

In this research, the solution of nonlinear optimal control problems using Artificial Neural Networks (ANN) has been presented. Since a closed loop configuration is better understood in its block diagram representation, the Block Partial Derivatives concept has facilitated the computation of the gradient. The algorithm has been adapted to solve tracking, regulation, terminal control, minimum control effort, minimum time, and output tracking problems. The techniques developed are of general nature since no a-priori assumptions are made about the performance index structure, the plant dynamics, or the problem-specific constrained. When the developed techniques have been applied to simulated plants, the results are found to closely agree to the open loop optimal solutions

obtained through other numerical techniques. The proposed technique has the added advantage of getting the control in closed loop form. The optimal control system has shown more robustness to change in the values of initial states and system parameters when the neural network is trained on a range of these values.

The techniques developed in this research do not guarantee convergence of the training process to a global minimum. One way to overcome this problem is to restart the training from different initial weights. Also, care should be taken while choosing values for the training parameters. Bad choice of the adaptation rate, for example, may cause the training to diverge. However, since the training is conducted off-line, the designer can try various values for these parameters. After satisfactory training and extensive test on the simulated plant, the controller can be implemented for the real plant.

5.2 Recommended Future Work

Following are four suggested extensions to this research work,

1. choice of the adaptation rate to ensure stability of the training algorithm,
2. a systematic approach to obtain the global minimum for the optimal control problem, for example, by augmenting our approach with another search method,
3. a comparison between the Radial Basis Network (RBN) and MFNN in optimal control,
4. replacing the MFNN by a neuro-fuzzy network where the human reasoning may be used in choosing decision parameters such as the initial network weights.

References

[1] Adeli, H. and H. Park, *Optimization of Space Structures by Neural Dynamics*, Neural Networks, Vol. 8, No. 5, 1995, pp. 769-781.

[2] Ahmed, M. and M. Anjum, *Learning Rate Algorithm for Neural Net Based Direct Adaptive Control*, The Arabian Journal for Science and Engineering, Vol. 18, No. 4, Oct. 1993, pp. 493-513.

[3] Ahmed, M., *Block Partial Derivative and its Application to Neural-Net-Based Direct-Model-Reference Adaptive Control*, IEE Proc.-Control Theory Appl., Vol. 141, No. 5, Sept. 1994, pp. 305-314.

[4] Ahmed, M., *BPD Computation and Model Reference Adaptive Control (MRAC) of Hammerstien Plants*, IEE-Proc.-Control Theory Appli. Vol. 142, No. 5, Sept. 1995, pp. 475-485.

[5] Antsaklis, P., *Neural Networks of Control Systems*, IEEE Trans., Neur., Net., Vol. 1, No. 2, Jun. 1990, pp. 242-244.

[6] Astrom, K. and B. Wittenmark, *Computer-Controlled System*, Prentice-Hall Inc., Englewood Cliffs, New Jersy, 1990.

[7] Balakrishnan, S. and V. Biega, *A New Neural Architecture for Homing Missile Guidance*. Proc. of the American Control Conf., Seattle, Washington, June 1995, pp. 2148-2152

[8] Baldi, P., *Gradient Descent Learning Algorithm Overview, A General Dynamical Systems Perspective*, IEEE Trans. Neur. Net., Vol. 6, No. 1, Jan. 1995, pp. 182-195.

[9] Biswas, S., *A Conjugate Hopfield Neural Network for Optimum System Control*, Proc. 29th IEEE Conf. Dec. Control, Honolulu, Hawaii, Dec. 1990, pp. 1757-1762.

[10] Bryson, A. and Y. Ho, *Applied Optimal Control*, Hemisphere Publ. Corp., Washington, D.C., 1975.

[11] Calise, *A Singular Perturbation Analysis of Optimal Aerodynamic and Thrust magnitude Control*, IEEE Trans. Aut. Control, AC-24, 1979, pp. 720.

[12] Guo and S. Gelfand, *Analysis of Gradient Descent Learning Algorithms for Multilayer Feed-Forward Neural Networks*, Proc. of the 20th Conf. on Dec. and Control, Honolulu, Hawaii, Dec. 1990, pp. 1751-1756.

[13] Haykin, S., *Neural Networks, A Comprehensive Foundation*, Macmillan Collage Pub. Company, New York, 1994.

[14] Holland, F. and F. Chapman, *Liquid Mixing and Processing in Stirred Tanks*, Reinhold Pub. Corp., New York, 1966.

[15] Hunt, K., D. Sbarbaro, R. Zbikowski, and P. Gawthrop, *Neural Networks for Control Systems-A survey*, Automatica, Vol. 28, No. 6, 1992, pp. 1083-1112.

[16] Ichikawa, Y., *Neural Networks Application for Direct Feedback Controllers*, IEEE Trans. Neur. Net., Vol. 3, No. 2, March 1992, pp. 224-231.

[17] Jain, A. and J. Mao, *Artificial Neural Networks: A Tutorial*, Computer, Vol. 29, No. 3, March 1996, pp. 31-44.

[18] Keerthi and E. Gilbert, *Moving-horizon approximations for a general class of optimal nonlinear infinite-horizon discrete-time systems*, In Proc. 20th Annual Conf. Information Science and Systems, Princeton University, pp. 301-306.

[19] Kirk, D., *Optimal Control Theory*, Prentice-Hall, Inc., New Jersey, 1970.

[20] Kwakernaak and R. Sivan, *Linear Optimal Control Systems*, Wiley-Interscience, New York, 1972, p. 123.

[21] Levin, A. and K. Narendra, *Control of Nonlinear Dynamical Systems using Neural Networks-Part II: Observability Identification, and Control*, IEEE Trans. Neur. Net., Vol. 7, No. 1, Jan. 1996, pp. 30-42.

[22] Lewis, F., *Optimal Control*, John Wiley and Sons, Inc., New York, 1986.

[23] Li, H. and C. Chen, *A Neural-Type network for Solving Minimimal Energy Path in Real Time*, IEEE Trans. Circ. Syst.-I, Vol. 40, No. 2, Feb. 1993, pp. 111-123.

[24] McDermott, W. and M. Athans, *Approximating Optimal Feedback Using Neural networks*, Proc. 33th IEEE Conf. Dec. Control, Lake Buena, FL, Dec. 1994. pp 2466-2471.

[25] Miller, W., R. Sutton, and P.J. Werbos, *Neural Networks for Control*, MIT Press, Cambridge, MA, pp. 59-65, 1990.

[26] Morari, M. and E. Zafiriou, *Robust Process Control*, Prentice-Hall, Englewood Cliffs, NJ, 1989.

[27] Narendra, K. and K. Parthasarathy, *Gradient Method for the Optimization of Dynamical Systems Containing Neural Networks*, IEEE Trans. Neur. Net. Vol. 2, Mar. 1991, pp. 252-262.

[28] Narendra, K. and K. Parthasarathy, *Identification and Control for Dynamical Systems Using Neural Networks*, IEEE Trans. Neur. Net., Vol. 1, No. 1, March 1990, pp. 4-27.

[29] Narendra, K. and S. Mukhopadhyay, *Neural Networks In Control Systems*, Proc. 31st IEEE Conf. Dec. Control, Tucson, Arizona, Dec. 1992, pp. 1-5.

[30] Nazim, A. and P. Shcherbakov, *NN-Approach to Design of the Optimal Stochastic Approximation Algorithms*, Proc. of the International Conf. on Control, Coventry, UK, Mar. 1994, pp. 449-453.

[31] Niesler, T. and J. Plessis, *Time-Optimal Control by Means of Neural Networks*, IEEE Control Systems, Oct. 1995, pp. 305-314.

[32] Omatu, M. Khalid, and R. Yusof, *Neuro-Control and its Applications*, Springer-Verlag London Limited, UK, 1996.

[33] Oppenheim, A. and R. Schafer, *Discrete-time Signal Processing*, Prentice-Hall Inc., 1989.

[34] Page, G., J. Gomm, and D. Williams, *Applic. of Neur. Networks to Modelling and Control*, Chapman and Hall, London, 1993.

[35] Parisini, T. and Zoppoli, *Neural Approximations for Multistage Optimal Control of Nonlinear Stochastic Systems*, IEEE Trans. Aut. Control, Vol. 41, No. 6, June 1996, pp. 889-895.

[36] Parisini, T. and R. Zoppoli, *Neural Optimal Control of Nonlinear Stochastic Systems*, Proc. of 1994 IEEE International Conf. on Neural Networks, Orlando, FL, June 1994, pp. 2383-2388.

[37] Park, Y., M. Choi, and K. Lee, *An Optimal Tracking Neuro_Controller for Nonlinear Dynamic Systems*, IEEE Transactions on Neural Networks, Vol. 7, No. 5, Sept. 1996, pp. 1099-1110.

[38] Plumer, E., *Optimal Control of Terminal Processes Using Neural Networks*, IEEE Trans. Neur. Net., Vol. 7, No. March 1996, pp. 408-417.

[39] Rumelhart, G. Hinton and R. Williams, *Learning Internal Representation by Back Propagation*, Parallel Distributed Processing: exploration in the microstructure of cognition, Vol. I, Cambridge, MA: Bradford books, 1986.

[40] Sheu, Vinh, and Howe, *Application of Singular Perturbation Methods for Three-Dimensional Minimum-time Interception*, Journal of Guidance, Control, and Dynamics, Vol. 14, No. 2, March-April 1991, pp. 360-367.

[41] Shinar, *On Application of Singular Perturbation Techniques in Nonlinear Optimal Control*, Automatica, Vol. 19, No. 2, 1983, pp. 203-211.

[42] Steck, J. and S. Balakrishnan, *Use of Hopfield Neural Networks in Optimal Guidance*, IEEE Trans. on Aerospace and Electronic Systems, Vol. 30, No. 1, Jan. 1994 pp. 287-293.

[43] Timofeev, A. and A. Bogdanov, *Synthesizing Principles of Neural Controllers for Robots Optimal Control*, The Second International Symposium on Neuroformatics and Neurocomputers, Rostov on Don, Russia, Sept. 1995, pp. 189-193.

[44] Visser and Shinar, *A Highly Accurate Feedback Approximation for Horizontal Variable-Speed Interceptions*, Journal of Guidance, Control, and Dynamics, Vol. 9, No. 6, Nov.-Dec. 1986, pp. 691-698.

[45] Werbos, P., *Backpropagation Through Time: What It Does and How to Do It*, Proceedings of the IEEE, Vol. 78, No. 10, Oct. 1990, pp. 1550-1560.

[46] Werbos, P., *Overview of Designs and Capabilities, in Neural Networks for Control*, MIT Press, Cambridge, MA, pp. 59-65, 1990.

[47] Widrow, B. and M. Lehr, *30 Years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation*, Proc. IEEE, Vol. 78, No. 9, Sept. 1990, pp. 1415-1442.

[48] Yen, G., *Optimal Tracking Control in Flexible Pointing Structures*, 1995 IEEE International Conf. on Systems, Man and Cybernetics. Intelligent Systems for the 21st Century, Vancouver, BC, Canada, Oct. 1995, pp. 4440-4445.

[49] Yen, V., T. Liu, and D. Liu, *Neural-network-based Near-time-optimal Position Control Method for DC Motor Servosystems*, IEE Proc. Control Theory Appl., Vol. 142, No. 5, Sept. 1995, pp. 493-500.

[50] Zakrzewski, R. and R. Mohler, *Discrete-Time Optimal Control Using Continuous Neural Networks*, Proc. of International Conf. on Neural Networks, Perth, WA, Australia, Nov. 1995.